ADSP-21160 SHARC® DSP
Hardware Reference

Revision 4.0, June 2009

Part Number
82-001966-01

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106

ANALOG DEVICES

Copyright Information

©2009 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, SHARC, the SHARC logo, and EZ-ICE are registered trademarks of Analog Devices, Inc.

VisualDSP++ is a trademark of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

CONTENTS

INTRODUCTION

Purpose 1-1
Audience 1-1
Overview – Why Floating-Point DSP? 1-2
ADSP-21160 DSP Design Advantages 1-2
ADSP-21160 DSP Architecture Overview 1-7
    Processor Core 1-7
        Processing Elements 1-7
        Program Sequence Control 1-9
        Processor Internal Buses 1-11
    Processor Peripherals 1-12
        Dual-Ported Internal Memory (SRAM) 1-12
        External Port 1-13
        I/O Processor 1-14
    JTAG Port 1-16
Development Tools 1-16
Differences From Previous SHARC DSPs 1-19
    Processor Core Enhancements 1-19
    Processor Internal Bus Enhancements 1-20

Memory Organization Enhancements 1-20
External Port Enhancements 1-21
Host Interface Enhancements 1-21
Multiprocessor Interface Enhancements 1-21
IO Architecture Enhancements 1-22
DMA Controller Enhancements 1-22
Link Port Enhancements 1-22
Instruction Set Enhancements 1-22
For Information About Analog Products 1-23
For Technical or Customer Support 1-24
What's New in This Manual 1-24
Related Documents 1-24
Conventions 1-25

PROCESSING ELEMENTS

Overview 2-1
Setting Computational Modes 2-2
32-bit (Normal Word) Floating-Point Format 2-3
40-bit Floating-Point Format 2-4
16-bit (Short Word) Floating-Point Format 2-5
32-Bit Fixed-Point Format 2-5
Rounding Mode 2-6
Using Computational Status 2-7
Arithmetic Logic Unit (ALU) 2-7
ALU Operation 2-8

ALU Saturation 2-9
ALU Status Flags 2-9
ALU Instruction Summary 2-10
Multiply—Accumulator (Multiplier) 2-13
Multiplier Operation 2-14
Multiplier (Fixed-Point) Result Register 2-15
Multiplier Status Flags 2-17
Multiplier Instruction Summary 2-18
Barrel-Shifter (Shifter) 2-21
Shifter Operation 2-21
Shifter Status Flags 2-26
Shifter Instruction Summary 2-27
Data Register File 2-29
Alternate (Secondary) Data Registers 2-31
Multifunction Computations 2-32
Secondary Processing Element (PEy) 2-36
Dual Compute Units Sets 2-38
Dual Register Files 2-39
Dual Alternate Registers 2-40
SIMD (Computational) Operations 2-40
SIMD and Status Flags 2-43

PROGRAM SEQUENCER

Overview 3-1
Instruction Pipeline 3-9

Instruction Cache 3-10
Using the Cache 3-12
Optimizing Cache Usage 3-13
Branches and Sequencing 3-15
Conditional Branches 3-17
Delayed Branches 3-17
Loops and Sequencing 3-21
Restrictions On Ending Loops 3-23
Restrictions On Short Loops 3-24
Loop Address Stack 3-29
Loop Counter Stack 3-30
Interrupts and Sequencing 3-33
Sensing Interrupts 3-40
Masking Interrupts 3-41
Latching Interrupts 3-42
Stacking Status During Interrupts 3-44
Nesting Interrupts 3-45
Reusing Interrupts 3-47
Interrupting IDLE 3-48
Multiprocessing Interrupts 3-49
Timer and Sequencing 3-50
Stacks and Sequencing 3-52
Conditional Sequencing 3-54
SIMD Mode and Sequencing 3-57

Conditional Compute Operations 3-58
Conditional Branches and Loops 3-59
Conditional Data Moves 3-59
Case 1:
Complementary Register Pair Data Move 3-59
Case 2:
Uncomplemented to Complementary Register Move 3-63
Case 3:
Complementary Register => Uncomplimentary Register . 3-64
Case 4:
Data Move Involves External Memory or IOP Memory Space 3-65
Conditional DAG Operations 3-66

DATA ADDRESS GENERATORS

Overview 4-1
Setting DAG Modes 4-3
Circular Buffering Mode 4-4
Broadcast Loading Mode 4-5
Alternate (Secondary) DAG Registers 4-6
Bit-Reverse Addressing Mode 4-8
Using DAG Status 4-9
DAG Operations 4-9
Addressing With DAGs 4-10
Addressing Circular Buffers 4-12
Modifying DAG Registers 4-18

Addressing in SISD and SIMD Modes 4-19
DAGs, Registers, and Memory 4-19
DAG Register-to-Bus Alignment 4-20
DAG Register Transfer Restrictions 4-22
DAG Instruction Summary 4-23

MEMORY

Overview 5-1
Internal Address and Data Buses 5-4
Internal Data Bus Exchange 5-7
ADSP-21160 DSP Memory Map 5-12
Internal Memory 5-14
Multiprocessor Memory 5-17
External Memory 5-20
Shadow Write FIFO 5-21
Memory Organization and Word Size 5-22
Placing 32-Bit Words and 48-Bit Words 5-23
Mixing 32-Bit and 48-Bit Words 5-23
Restrictions on Mixing 32-Bit and 48-Bit Words 5-26
Setting Data Access Modes 5-28
Using Boot Memory 5-29
Reading from Boot Memory 5-30
Writing to Boot Memory 5-31
Internal Interrupt Vector Table 5-31
Internal Memory Block Data Width 5-32

Memory Bank Size 5-33
External Bus Priority 5-33
Secondary Processor Element (PEy) 5-34
Broadcast Register Loads 5-34
Illegal I/O Processor Register Access 5-35
Unaligned 64-bit Memory Access 5-36
External Bank X Access Mode 5-36
External Bank X Waitstates 5-37
External (Bank 0) DRAM Page Size 5-38
Using Memory Access Status 5-38
Accessing Memory 5-39
Access Word Size 5-40
Long Word (64-Bit) Accesses 5-41
Instruction Word (48-Bit) and Extended Precision Normal Word (40-Bit) Accesses 5-43
Normal Word (32-Bit) Accesses 5-43
Short Word (16-Bit) Accesses 5-44
SISD, SIMD, and Broadcast Load Modes 5-44
Single-and Dual-Data Accesses 5-45
Data Access Options 5-45
Short Word Addressing of Single Data in SISD Mode 5-47
Short Word Addressing of Single Data in SIMD Mode 5-48
Short Word Addressing of Dual-Data in SISD Mode 5-51
Short Word Addressing of Dual-Data in SIMD Mode 5-53
32-Bit Normal Word Addressing of Single Data in SISD Mode 5-55

32-Bit Normal Word Addressing of Single Data in SIMD Mode 5-57
32-Bit Normal Word Addressing of Dual Data in SISD Mode 5-59
32-Bit Normal Word Addressing of Dual Data in SIMD Mode 5-59
Extended Precision Normal Word Addressing of Single Data 5-62
Extended Precision Normal Word Addressing of Dual Data in SISD Mode 5-64
Extended Precision Normal Word Addressing of Dual Data in SIMD Mode 5-64
Long Word Addressing of Single Data 5-67
Long Word Addressing of Dual Data in SISD Mode 5-69
Long Word Addressing of Dual Data in SIMD Mode 5-69
Mixed Word Width Addressing of Dual Data in SISD Mode 5-72
Mixed Word Width Addressing of Dual Data in SIMD Mode 5-74
Broadcast Load Access 5-74
Arranging Data in Memory 5-84

I/O PROCESSOR

Overview 6-1
Setting I/O Processor—EPort Modes 6-13
Boot Memory DMA Mode 6-16
External Port Buffer Modes 6-17
External Port Channel Priority Modes 6-18
External Port Channel Transfer Modes 6-21
External Port Channel Handshake Modes 6-22
Master Mode 6-24

Paced Master Mode 6-29
Slave Mode 6-30
Handshake Mode 6-33
External-Handshake Mode 6-40
Setting I/O Processor—LPort Modes 6-41
Link Port Buffer Modes 6-44
Link Port Channel Priority Modes 6-45
Link Port Channel Transfer Modes 6-47
Setting I/O Processor—SPort Modes 6-49
Serial Port Buffer Modes 6-51
Serial Port Channel Priority Modes 6-52
Serial Port Channel Transfer Modes 6-52
Using I/O Processor Status 6-53
External Port Status 6-57
Link Port Status 6-60
Serial Port Status 6-63
DMA Controller Operation 6-65
Managing DMA Channel Priority 6-66
Chaining DMA Processes 6-70
Transfer Control Block (TCB) Chain Loading 6-71
Setting Up and Starting The Chain 6-73
Inserting a TCB in an Active Chain 6-73
External Port DMA 6-75
Setting up External Port DMA 6-75

Bootloading Through The External Port 6-76
Link Port DMA 6-81
Setting up Link Port DMA 6-81
Using Two-Dimensional Link Port DMA 6-83
Bootloading Through The Link Port 6-88
Serial Port DMA 6-91
Setting up Serial Port DMA 6-91
Using Two-Dimensional Serial Port DMA 6-93
Optimizing DMA Throughput 6-93
Internal Memory DMA 6-93
External Memory DMA 6-94
System-Level Considerations 6-97

EXTERNAL PORT

Overview 7-1
Setting External Port Modes 7-2
External Memory Interface 7-3
Banked External Memory 7-10
Unbanked External Memory 7-10
Boot Memory 7-11
Idle Cycle 7-11
Data Hold Cycle 7-13
Multiprocessor Memory Space Waitstates and Acknowledge 7-14
DRAM Page Boundary Detection 7-15
Timing External Memory Accesses 7-18

Asynchronous Mode Interface Timing 7-19
Asynchronous Mode Read – Bus Master 7-21
Asynchronous Mode Write – Bus Master 7-22
Synchronous Mode Interface Timing 7-23
Synchronous Mode Read – Bus Master 7-25
Synchronous Write, Zero-Waitstate Mode 7-28
Synchronous Write, One Waitstate Mode 7-32
Synchronous Burst Mode Interface Timing 7-34
Burst Length Determination 7-36
Burst Stall Criteria 7-36
Synchronous Burst Reads 7-38
Synchronous Burst Writes 7-40
Using External SBSRAM 7-44
Executing Instructions From External Memory 7-49
Host Processor Interface 7-51
Acquiring the Bus 7-55
Asynchronous Transfers 7-59
Asynchronous Transfer Timing 7-61
Synchronous Transfers 7-64
Synchronous Broadcast Writes 7-65
Synchronous Burst Read Transfers 7-67
Slave Direct Reads and Writes 7-68
IOP Shadow Registers 7-68
Instruction Transfers 7-69

Host Direct Writes and Reads 7-70
Direct Writes 7-70
Direct Write Latency 7-71
Direct Reads 7-72
Broadcast Writes 7-73
Shadow Write FIFO 7-73
Data Transfers Through the EPBx Buffers 7-74
DMA Transfers 7-74
Host Data Packing 7-75
32-bit Data Packing 7-75
48-Bit Instruction Packing 7-79
Host Interface Status 7-81
Interprocessor Messages and Vector Interrupts 7-81
Message Passing (MSGRx) 7-82
Host Vector Interrupts (VIRPT) 7-82
System Bus Interfacing 7-83
Access to the DSP Bus—Slave DSP 7-84
Access to the System Bus—Master DSP 7-84
Processor Core Access To System Bus 7-87
Deadlock Resolution 7-87
DSP DMA Access To System Bus 7-91
Multiprocessing with Local Memory 7-92
DSP To Microprocessor Interface 7-94
Multiprocessor (DSPs) Interface 7-94

Multiprocessing System Architectures 7-97
Data Flow Multiprocessing 7-97
Cluster Multiprocessing 7-98
Multiprocessor Bus Arbitration 7-101
Bus Arbitration Protocol 7-103
Bus Arbitration Priority (RPBA) 7-107
Mastership Timeout Bus 7-108
Priority Access 7-110
Bus Synchronization After Reset 7-113
Booting Another DSP 7-116
Multiprocessor Direct Writes and Reads 7-116
IOP Shadow Registers 7-117
Instruction Transfers 7-117
Direct Writes 7-117
Direct Write Latency 7-118
Direct Reads 7-118
Broadcast Writes 7-118
Shadow Write FIFO 7-120
Data Transfers Through the EPBx Buffers 7-121
Bus Lock and Semaphores 7-121
Interprocessor Messages and Vector Interrupts 7-123
Message Passing (MSGRx) 7-124
Vector Interrupts (VIRPT) 7-124
Multiprocessor Interface Status 7-125

LINK PORTS

Overview 8-1
Link Port To Link Buffer Assignment 8-3
Link Port DMA Channels 8-5
Link Port Booting 8-5
Setting Link Port Modes 8-5
Link Data Path (and Compatibility) Modes 8-7
Using Link Port Handshake Signals 8-8
Using Link Buffers 8-10
Core Processor Access To Link Buffers 8-11
Host Processor Access To Link Buffers 8-12
Using Link Port DMA 8-12
Using Link Port Interrupts 8-12
Link Port Interrupts With DMA Enabled 8-13
Link Port Interrupts With DMA Disabled 8-14
Link Port Service Request Interrupts (LSRQ) 8-14
Detecting Errors On Link Transmissions 8-16
Using Token Passing With Link Ports 8-17
Designing Link Port Systems 8-21
Terminations For Link Transmission Lines 8-21
Peripheral I/O Using Link Ports 8-21
Data Flow Multiprocessing With Link Ports 8-22

SERIAL PORTS

Overview 9-1
SPORT Interrupts 9-5
SPORT Reset 9-6
Setting Serial Port Modes 9-7
Transmit and Receive Control Registers (STCTL, SRCTL) 9-9
Register Writes and Effect Latency 9-12
Transmit and Receive Data Buffers (TX, RX) 9-12
Clock and Frame Sync Frequencies (TDIV, RDIV) 9-14
Data Word Formats 9-17
Word Length 9-17
Endian Format 9-18
Data Packing and Unpacking 9-18
Data Type 9-19
Companding 9-20
Clock Signal Options 9-21
Frame Sync Options 9-22
Framed Versus Unframed 9-22
Internal Versus External Frame Syncs 9-24
Active Low Versus Active High Frame Syncs 9-24
Sampling Edge For Data and Frame Syncs 9-25
Early Versus Late Frame Syncs 9-25
Data-Independent Transmit Frame Sync 9-27

SPORT Loopback 9-27
Multichannel Operation 9-28
Frame Syncs in Multichannel Mode 9-30
Multichannel Control Bits in STCTL, SRCTL 9-30
Channel Selection Registers 9-32
SPORT Receive Comparison Registers 9-33
Moving Data Between SPORTS and Memory 9-36
DMA Block Transfers 9-36
Single-Word Transfers 9-37
SPORT Pin/Line Terminations 9-38

JTAG TEST EMULATION PORT

JTAG Test Access Port 10-3
Instruction Register 10-4
EMUPMD Shift Register 10-5
EMUPX Shift Register 10-6
EMU64PX Shift Register 10-7
EMUPC Shift Register 10-7
EMUCTL Shift Register 10-7
EMUSTAT Shift Register 10-11
BRKSTAT Shift Register 10-12
MEMTST Shift Register 10-13
PSx, DMx, IOx, and EPx (Breakpoint) Registers 10-13
EMUN Register 10-16
EMUCLK and EMUCLK2 Registers 10-16

EMUIDLE Instruction 10-17
In-Circuit Signal Analyzer (ICSA) Function 10-17
Boundary Register 10-17
Device Identification Register 10-37
Built-in Self-test Operation (BIST) 10-37
Private Instructions 10-37
References 10-38

SYSTEM DESIGN

DSP Pin Descriptions 11-1
Pin States At Reset 11-12
Clock Derivation 11-14
RESET and CLKIN 11-16
Input Synchronization Delay 11-17
Interrupt and Timer Pins 11-18
Flag Pins 11-18
Flag Inputs 11-19
Flag Outputs 11-19
JTAG Interface Pins 11-20
Dual-Voltage Power-up Sequencing 11-21
Designing for JTAG Emulation 11-24
Target Board Connector 11-25
Layout Requirements 11-30
Power Sequence for Emulation 11-31
Additional JTAG Emulator References 11-31

Pod Specifications 11-32
DSP JTAG Pod Connector 11-32
DSP 3.3V Pod Logic 11-33
DSP 2.5V Pod Logic 11-34
Conditioning Input Signals 11-35
Link Port Input Filter Circuits 11-35
RESET Input Hysteresis 11-36
Designing For High Frequency Operation 11-36
Clock Specifications and Jitter 11-37
Clock Distribution 11-38
Point-To-Point Connections 11-40
Signal Integrity 11-41
Other Recommendations and Suggestions 11-44
Decoupling Capacitors and Ground Planes 11-45
Oscilloscope Probes 11-47
Recommended Reading 11-47
Booting Single and Multiple Processors 11-48
Multiprocessor Host Booting 11-48
Multiprocessor EPROM Booting 11-49
All DSPs Boot in Turn from a Single EPROM 11-49
One DSP is Booted, which then Boots the Others 11-50
Multiprocessor Link Port Booting 11-51
Multiprocessor Booting From External Memory 11-52

REGISTERS

Control and Status System Registers A-2
Mode Control 1 Register (MODE1) A-3
Mode Mask Register (MMASK) A-5
Mode Control 2 Register (MODE2) A-6
Arithmetic Status Registers (ASTATx and ASTATy) A-8
Sticky Status Registers (STKYx and STKYy) A-8
User-Defined Status Registers (USTATx) A-15
Processing Element Registers A-15
Data File Data Registers (Rx, Fx, Sx) A-15
Multiplier Results Registers (MRxF, MRxB) A-16
Program Memory Bus Exchange Register (PX) A-16
Program Sequencer Registers A-17
Interrupt Latch Register (IRPTL) A-18
Interrupt Mask Register (IMASK) A-22
Interrupt Mask Pointer Register (IMASKP) A-23
Link Port Interrupt Register (LIRPTL) A-24
Flag Value Register (FLAGS) A-27
Program Counter Register (PC) A-28
Program Counter Stack Register (PCSTK) A-29
Program Counter Stack Pointer Register (PCSTKP) A-29
Fetch Address Register (FADDR) A-30
Decode Address Register (DADDR) A-30
Loop Address Stack Register (LADDR) A-30

Current Loop Counter Register (CURLCNTR) A-31
Loop Counter Register (LCNTR) A-31
Timer Period Register (TPERIOD) A-31
Timer Count Register (TCOUNT) A-31
Data Address Generator Registers A-31
Index Registers (Ix) A-32
Modify Registers (Mx) A-32
Length and Base Register (Lx, Bx) A-33
I/O Processor Registers A-33
System Configuration Register (SYSCON) A-45
Vector Interrupt Address Register (VIRPT) A-47
External Memory Waitstate and Access Mode Register (WAIT) A-48
System Status Register (SYSTAT) A-51
External Port DMA Buffer Registers (EPBx) A-52
Message Registers (MSGRx) A-53
PC Shadow Register (PC_SHDW) A-53
MODE2 Shadow Register (MODE2_SHDW) A-53
Bus Timeout Maximum Register (BMAX) A-53
Bus (Timeout) Counter Register (BCNT) A-54
Address of Last DRAM Page Register (ELAST) A-54
External Port DMA Control Registers (DMACx) A-54
Internal Memory DMA Index Registers (IIx) A-58
Internal Memory DMA Modifier Registers (IMx) A-59
Internal Memory DMA Count Registers (Cx) A-59

Chain Pointer For Next DMA TCB Registers (CPx) A-59
General Purpose DMA Registers (GPx, DBx, DAx) A-60
DMA Channel Status Register (DMASTAT) A-60
External Memory DMA Index Registers (EIx) A-61
External Memory DMA Modifier Registers (EMx) A-61
External Memory DMA Count Registers (ECx) A-62
Link Port Buffer Registers (LBUFx) A-62
Link Port Buffer Control Registers (LCTLx) A-62
Link Port Common Control Register (LCOM) A-65
Link Port Assignment Register (LAR) A-67
Link Port Service Request and Mask Register (LSRQ) A-68
Link Port Path Registers (LPATHx) A-70
Link Port Path Counter Register (LPCNT) A-70
Link Port Constant Registers (CNSTx) A-71
SPORT Serial Transmit Control Registers (STCTLx) A-71
SPORT Serial Receive Control Registers (SRCTLx) A-73
SPORT Transmit Buffer Registers (TXx) A-73
SPORT Receive Buffer Registers (RXx) A-76
SPORT Transmit Divisor Registers (TDIVx) A-77
SPORT Transmit Count Registers (TCNTx) A-78
SPORT Receive Divisor Registers (RDIVx) A-78
SPORT Receive Count Registers (RCNTx) A-79
SPORT Transmit Select Registers (MTCSx) A-79
SPORT Receive Select Registers (MRCSx) A-80

SPORT Transmit Compand Registers (MTCCSx) A-80
SPORT Receive Compand Register (MRCCSx) A-80
SPORT Receive Comparison and Mask Registers (KEYWDx and KEYMASKx) A-81
SPORT Serial Path Length Registers (SPATHx) A-81
SPORT Serial Path Counter Registers (SPCNTx) A-82
Register and Bit #Defines File (def21160.h) A-82

INTERRUPT VECTOR ADDRESSES

NUMERIC FORMATS

IEEE Single-Precision Floating-Point Data Format C-1
Extended-Precision Floating-Point Format C-3
Short Word Floating-Point Format C-4
Packing for Floating-Point Data C-4
Fixed-Point Formats C-6

GLOSSARY

INDEX

1 INTRODUCTION

Thank you for purchasing Analog Devices SHARC® digital signal processor (DSP).

Purpose

The ADSP-21160 SHARC DSP Hardware Reference provides architectural information on the ADSP-21160 Super Harvard Architecture (SHARC) Digital Signal Processor (DSP). The architectural descriptions cover functional blocks, busses, and ports, including all features and processes they support. For programming information, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Audience

DSP system designers and programmers who are familiar with signal processing concepts are the primary audience for this manual. This manual assumes that the audience has a working knowledge of microcomputer technology and DSP-related mathematics.

DSP system designers and programmers who are unfamiliar with signal processing can use this manual, but should supplement this manual with other texts, describing DSP techniques.

All readers, particularly system designers, should refer to the DSP's data sheet for timing, electrical, and package specifications. For additional suggested reading, see "For Information About Analog Products" on page 1-23.

Overview – Why Floating-Point DSP?

A digital signal processor's data format determines its ability to handle signals of differing precision, dynamic range, and signal-to-noise ratios. Because floating-point DSP math reduces the need for scaling and probability of overflow, using a floating-point DSP can ease algorithm and software development. The extent to which this is true depends on the floating-point processor's architecture. Consistency with IEEE workstation simulations and the elimination of scaling are two clear ease-of-use advantages. High-level language programmability, large address spaces, and wide dynamic range allow system development time to be spent on algorithms and signal processing concerns, rather than assembly language coding, code paging, and error handling. The ADSP-21160 is a highly integrated, 32-bit floating-point DSP that provides many of these design advantages.

ADSP-21160 DSP Design Advantages

The ADSP-21160 processor is a high-performance 32-bit DSP for medical imaging, communications, military, audio, test equipment, 3D graphics, speech recognition, motor control, imaging, and other applications. This DSP builds on the ADSP-21000 DSP core to form a complete system-on-a-chip, adding a dual-ported on-chip SRAM, integrated I/O peripherals, and an additional processing element for Single-Instruction-Multiple-Data (SIMD) support.

SHARC is an acronym for Super Harvard Architecture. This DSP architecture balances a high performance processor core with high performance

buses (PM, DM, IO). In the core, every instruction can execute in a single cycle. The buses and instruction cache provide rapid, unimpeded data flow to the core to maintain the execution rate.

Figure 1-1 shows a detailed block diagram of the processor, illustrating the following architectural features:

- Two Processing Elements (PEx and PEy), each containing a 32-Bit IEEE floating-point computation units—multiplier, ALU, Shifter, and data register file
- Program sequencer with related instruction cache, interval timer, and Data Address Generators (DAG1 and DAG2)
- Dual-ported SRAM
- External port for interfacing to off-chip memory, peripherals, hosts, and multiprocessor systems
- Input/Output (IO) processor with integrated DMA controller, serial ports, and link ports for point-to-point multiprocessor communications
- JTAG Test Access Port for emulation

Figure 1-1 also shows the three on-chip buses of the ADSP-21160: the Program Memory (PM) bus, Data Memory (DM) bus, and Input/Output (IO) bus. The PM bus provides access to either instructions or data. During a single cycle, these buses let the processor access two data operands (one from PM and one from DM), access an instruction (from the cache), and perform a DMA transfer.

The buses connect to the ADSP-21160 DSP's external port, which provides the processor's interface to external memory, memory-mapped I/O, a host processor, and additional multiprocessing ADSP-21160 DSPs. The

external port performs bus arbitration and supplies control signals to shared, global memory and I/O devices.

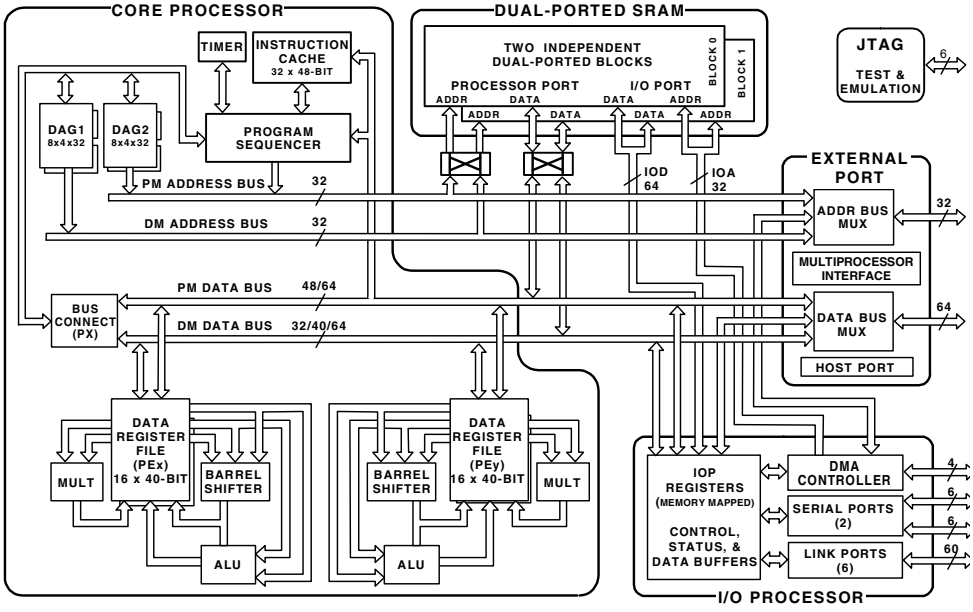


Figure 1-1. ADSP-21160 SHARC DSP Block Diagram

Figure 1-2 illustrates a typical single-processor system. The ADSP-21160 DSP includes extensive support for multiprocessor systems as well. For more information, see "Multiprocessor (DSPs) Interface" on page 7-94.

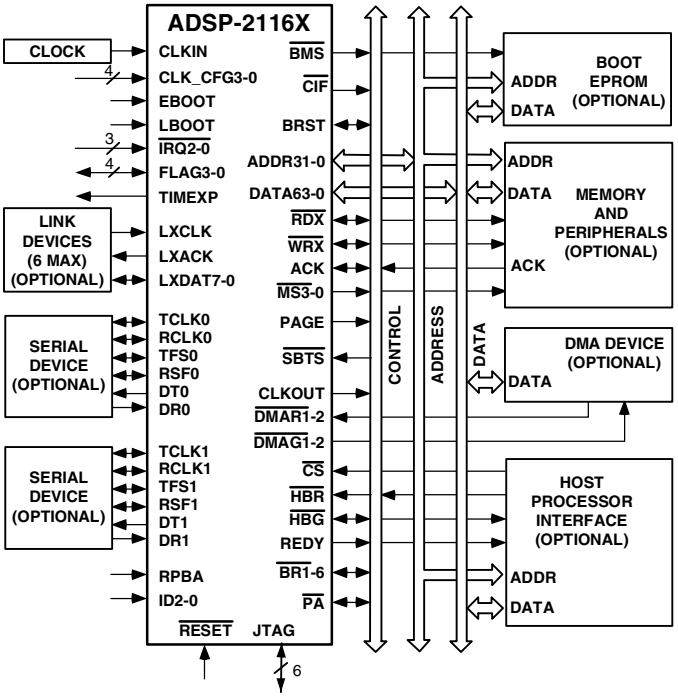


Figure 1-2. ADSP-21160 Processor System

Further, the ADSP-21160 DSP addresses the five central requirements for DSPs:

- Fast, flexible arithmetic computation units
- Unconstrained data flow to and from the computation units
- Extended precision and dynamic range in the computation units

- Dual address generators with circular buffering support
- Efficient program sequencing

Fast, Flexible Arithmetic. The ADSP-21000 Family processors execute all instructions in a single cycle. They provide both fast cycle times and a complete set of arithmetic operations. The DSP is IEEE floating-point compatible and allows either interrupt on arithmetic exception or latched status exception handling.

Unconstrained Data Flow. The ADSP-21160 DSP has a Super Harvard Architecture combined with a 10-port data register file. In every cycle, the DSP can write or read two operands to or from the register file, supply two operands to the ALU, supply two operands to the multiplier, and receive three results from the ALU and multiplier. The processor's 48-bit orthogonal instruction word supports parallel data transfers and arithmetic operations in the same instruction.

40-Bit Extended Precision. The DSP handles 32-bit IEEE floating-point format, 32-bit integer and fractional formats (twos-complement and unsigned), and extended-precision 40-bit floating-point format. The processors carry extended precision throughout their computation units, limiting intermediate data truncation errors.

Dual Address Generators. The DSP has two data address generators (DAGs) that provide immediate or indirect (pre- and post-modify) addressing. Modulus, bit-reverse, and broadcast operations are supported with no constraints on data buffer placement.

Efficient Program Sequencing. In addition to zero-overhead loops, the DSP supports single-cycle setup and exit for loops. Loops are both nestable (six levels in hardware) and interruptable. The processors support both delayed and non-delayed branches

ADSP-21160 DSP Architecture Overview

The ADSP-21160 DSP forms a complete system-on-a-chip, integrating a large, high-speed SRAM and I/O peripherals supported by a dedicated I/O bus. The following sections summarize the features of each functional block in the ADSP-21160 SHARC architecture, which appears in Figure 1-1 on page 1-4. With each summary, a cross reference points to the sections where the features are described in greater detail.

Processor Core

The processor core of the ADSP-21160 DSP consists of two processing elements (each with three computation units and data register file), a program sequencer, two data address generators, a timer, and an instruction cache. All digital signal processing occurs in the processor core.

Processing Elements

The processor core contains two Processing Elements (PEx and PEy). Each element contains a data register file and three independent computation units: an ALU, a multiplier with a fixed-point accumulator, and a shifter. For meeting a wide variety of processing needs, the computation units process data in three formats: 32-bit fixed-point, 32-bit floating-point and 40-bit floating-point. The floating-point operations are single-precision IEEE-compatible. The 32-bit floating-point format is the standard IEEE format, whereas the 40-bit extended-precision format has eight additional Least Significant Bits (LSBs) of mantissa for greater accuracy.

The ALU performs a set of arithmetic and logic operations on both fixed-point and floating-point formats. The multiplier performs floating-point or fixed-point multiplication and fixed-point multiply/add or multiply/subtract operations. The shifter performs logical and arithmetic shifts, bit manipulation, field deposit and extraction, and exponent derivation operations on 32-bit operands.

These computation units perform single-cycle operations; there is no computation pipeline. All units are connected in parallel, rather than serially. The output of any unit may serve as the input of any unit on the next cycle. In a multifunction computation, the ALU and multiplier perform independent, simultaneous operations.

Each processing element has a general-purpose data register file that transfers data between the computation units and the data buses and stores intermediate results. A register file has two sets (primary and alternate) of sixteen registers each, for fast context switching. All of the registers are 40 bits wide. The register file, combined with the core processor's Harvard architecture, allows unconstrained data flow between computation units and internal memory.

Primary Processing Element (PEx). PEx processes all computational instructions whether the DSP is in Single-Instruction, Single-Data (SISD) or Single-Instruction, Multiple-Data (SIMD) mode. This element corresponds to the computational units and register file in previous ADSP-21000 DSPs.

Secondary Processing Element (PEy). PEy processes each computational instruction in lock-step with PEx, but only processes these instructions when the DSP is in SIMD mode. Because many operations are influenced by this mode, more information on SIMD is available in multiple locations:

- For information on PEy operations, see "Processing Elements"
- For information on data addressing in SIMD mode, see "Addressing in SISD and SIMD Modes" on page 4-19
- For information on data accesses in SIMD mode, see "SISD, SIMD, and Broadcast Load Modes" on page 5-44

- For information on multiprocessing in SIMD mode, see "Multiprocessor (DSPs) Interface" on page 7-94.
- For information on SIMD programming, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Program Sequence Control

Internal controls for ADSP-21160 DSP's program execution come from four functional blocks: program sequencer, data address generators, timer, and instruction cache. Two dedicated address generators and a program sequencer supply addresses for memory accesses. Together the sequencer and data address generators allow computational operations to execute with maximum efficiency since the computation units can be devoted exclusively to processing data. With its instruction cache, the ADSP-21160DSP can simultaneously fetch an instruction from the cache and access two data operands from memory. The data address generators implement circular data buffers in hardware.

Program Sequencer. The program sequencer supplies instruction addresses to program memory. It controls loop iterations and evaluates conditional instructions. With an internal loop counter and loop stack, the ADSP-21160 DSP executes looped code with zero overhead. No explicit jump instructions are required to loop or decrement and test the counter.

The ADSP-21160 DSP achieves its fast execution rate by means of pipelined fetch, decode and execute cycles. If external memories are used, they are allowed more time to complete an access than if there were no decode cycle.

Data Address Generators. The data address generators (DAGs) provide memory addresses when data is transferred between memory and registers. Dual data address generators enable the processor to output simultaneous addresses for two operand reads or writes. DAG1 supplies 32-bit addresses

to data memory. DAG2 supplies 32-bit addresses to program memory for program memory data accesses.

Each DAG keeps track of up to eight address pointers, eight modifiers and eight length values. A pointer used for indirect addressing can be modified by a value in a specified register, either before (pre-modify) or after (post-modify) the access. A length value may be associated with each pointer to perform automatic modulo addressing for circular data buffers; the circular buffers can be located at arbitrary boundaries in memory. Each DAG register has an alternate register that can be activated for fast context switching.

Circular buffers allow efficient implementation of delay lines and other data structures required in digital signal processing, and are commonly used in digital filters and Fourier transforms. The DAGs automatically handle address pointer wraparound, reducing overhead, increasing performance, and simplifying implementation.

Interrupts. The ADSP-21160 DSP has four external hardware interrupts: three general-purpose interrupts, $\overline{\texttt{IRQ2-0}}$, and a special interrupt for reset. The processor also has internally generated interrupts for the timer, DMA controller operations, circular buffer overflow, stack overflows, arithmetic exceptions, multiprocessor vector interrupts, and user-defined software interrupts.

For the general-purpose external interrupts and the internal timer interrupt, the ADSP-21160 DSP automatically stacks the arithmetic status and mode (MODE1) registers in parallel with the interrupt servicing, allowing fifteen nesting levels of very fast service for these interrupts.

Context Switch. Many of the processor's registers have alternate registers that can be activated during interrupt servicing for a fast context switch. The data registers in the register file, the DAG registers, and the multiplier result register all have alternates. The Primary Registers are active at reset, while the Alternate (or Secondary) Registers are activated by control bits in a mode control register.

Timer. The programmable interval timer provides periodic interrupt generation. When enabled, the timer decrements a 32-bit count register every cycle. When this count register reaches zero, the ADSP-21160 DSP generates an interrupt and asserts its timer expired output. The count register is automatically reloaded from a 32-bit period register and the count resumes immediately.

Instruction Cache. The program sequencer includes a 32-word instruction cache that enables three-bus operation for fetching an instruction and two data values. The cache is selective—only instructions whose fetches conflict with program memory data accesses are cached. This caching allows full-speed execution of core, looped operations such as digital filter multiply-accumulates and FFT butterfly processing.

Processor Internal Buses

The processor core has six buses: PM address, PM data, DM address, DM data, IO address, and IO data. Due to processor's Harvard Architecture, data memory stores data operands, while program memory can store both instructions and data. This architecture allows dual data fetches, when the instruction is supplied by the cache.

Bus Capacities. The PM address bus and DM address bus transfer the addresses for instructions and data. The PM data bus and DM data bus transfer the data or instructions from each type of memory. The PM address bus is 32 bits wide allowing access of up to 4 Gwords of mixed instructions and data. The PM data bus is 64 bits wide to accommodate the 48-bit instructions and 64-bit data.

The DM address bus is 32 bits wide allowing direct access of up to 4G words of data. The DM data bus is 64 bits wide. The DM data bus provides a path for the contents of any register in the processor to be transferred to any other register or to any data memory location in a single cycle. The data memory address comes from one of two sources: an absolute value specified in the instruction code (direct addressing) or the output of a data address generator (indirect addressing).

The IO address and IO data buses let the IO processor access internal memory for DMA without delaying the processor core. The IO address bus is 32 bits wide, and the IO data bus is 64 bits wide.

Data Transfers. Nearly every register in the processor core is classified as a Universal Register (UREG). Instructions allow transferring data between any two universal registers or between a universal register and memory. This support includes transfers between control registers, status registers, and data registers in the register file. The PM bus connect (PX) registers permit data to be passed between the 64-bit PM data bus and the 64-bit DM data bus or between the 40-bit register file and the PM data bus. These registers contain hardware to handle the data width difference. For more information, see "Processing Element Registers" on page A-15.

Processor Peripherals

The term processor peripherals refers to everything outside the processor core. The ADSP-21160 DSP's peripherals include internal memory, external port, I/O processor, JTAG port, and any external devices that connect to the DSP.

Dual-Ported Internal Memory (SRAM)

The ADSP-21160 DSP contains 4 megabits of on-chip SRAM, organized as two blocks of 2 Mbits each, which can be configured for different combinations of code and data storage. Each memory block is dual-ported for single-cycle, independent accesses by the core processor and I/O processor or DMA controller. The dual-ported memory and separate on-chip buses allow two data transfers from the core and one from I/O, all in a single cycle.

All of the memory can be accessed as 16-, 32-, 48-, or 64-bit words. On the ADSP-21160 DSP, the memory can be configured as a maximum of 128K words of 32-bit data, 256K words of 16-bit data, 80K words of

48-bit instructions (and 40-bit data), or combinations of different word sizes up to 4 megabits.

The DSP supports a 16-bit floating-point storage format, which effectively doubles the amount of data that may be stored on chip. Conversion between the 32-bit floating-point and 16-bit floating-point formats completes in a single instruction.

While each memory block can store combinations of code and data, accesses are most efficient when one block stores data, using the DM bus for transfers, and the other block stores instructions and data, using the PM bus for transfers. Using the DM bus and PM bus in this way, with one dedicated to each memory block, assures single-cycle execution with two data transfers. In this case, the instruction must be available in the cache. The DSP also maintains single-cycle execution when one of the data operands is transferred to or from off-chip, using the DSP's external port.

External Port

The ADSP-21160 DSP's external port provides the processor's interface to off-chip memory and peripherals. The 4-gigaword off-chip address space is included in the ADSP-21160 DSP's unified address space. The separate on-chip buses—for PM address, PM data, DM address, DM data, IO address, and IO data—multiplex at the external port to create an external system bus with a single 32-bit address bus and a single 64-bit data bus. External SRAM can be 16, 32, 48, or 64 bits wide; the DSP's on-chip DMA controller automatically packs external data into the appropriate word width during transfers.

On-chip decoding of high-order address lines generates memory bank select signals for addressing external memory devices. Separate control lines support simplified addressing of page-mode DRAM. The ADSP-21160 DSP provides programmable memory waitstates and external memory acknowledge controls for interfacing to DRAM and peripherals with variable access, hold, and disable time requirements.

Host Processor Interface. The ADSP-21160 DSP's host interface allows easy connection to standard microprocessor buses, both 16-bit and 32-bit, with little additional hardware required. The interface supports asynchronous and synchronous transfers at speeds up to the half the internal clock rate of the ADSP-21160 DSP. The host interface operates through the DSP's external port and maps into the unified address space. Four channels of DMA are available for the host interface; code and data transfers occur with low software overhead. The host can directly read and write the internal memory of the ADSP-21160 DSP and can access the DMA channel setup and mailbox registers. Vector interrupt support provides for efficient execution of host commands.

Multiprocessor System Interface. The ADSP-21160 DSP offers powerful features tailored to multiprocessing DSP systems. The unified address space allows direct interprocessor accesses of each ADSP-21160 DSP's internal memory. Distributed bus arbitration logic on the DSP allows simple, glueless connection of systems containing up to six ADSP-21160 DSPs and a host processor. Master processor changeover incurs only one cycle of overhead. Bus arbitration handles either fixed or rotating priority. Processor bus lock allows indivisible read-modify-write sequences for semaphores. A vector interrupt capability is provided for interprocessor commands. Broadcast writes allow simultaneous transmission of data to all ADSP-21160 DSPs and can be used to implement reflective semaphores.

I/O Processor

The ADSP-21160 DSP's Input/Output Processor (IOP) includes two serial ports, six link ports, and a DMA controller. One of the I/O processes that the IO processor automates is booting. The DSP can boot from the external port (with data from an 8-bit EPROM or a host processor) or a link port. Alternatively, a no-boot mode lets the DSP start by executing instructions from external memory without booting.

Serial Ports. The ADSP-21160 DSP features two synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices. The serial ports can operate at up to half the processor core clock rate. Independent transmit and receive functions provide greater flexibility for serial communications. Serial port data can automatically transfer to and from on-chip memory using DMA. Each of the serial ports offers a TDM multichannel mode and supports m-law or A-law companding.

The serial ports can operate with little-endian or big-endian transmission formats, with word lengths selectable from 3 to 32 bits. They offer selectable synchronization and transmit modes. Serial port clocks and frame syncs can be internally or externally generated.

Link Ports. The ADSP-21160 DSP features six 8-bit link ports that provide additional I/O capabilities. Link port I/O is especially useful for point-to-point interprocessor communication in multiprocessing systems. The link ports can operate independently and simultaneously. The data packs into 32-bit or 48-bit words, which the processor core can directly read or the IO processor can DMA-transfer to on-chip memory. Clock/acknowledge handshaking controls link port transfers. Transfers are programmable as either transmit or receive.

DMA Controller. The ADSP-21160 DSP's on-chip DMA controller allows zero-overhead data transfers without processor intervention. The DMA controller operates independently and invisibly to the processor core to enable DMA operations to occur while the core is simultaneously executing its program. Both code and data can be downloaded to the ADSP-21160 DSP using DMA transfers.

DMA transfers can occur between the ADSP-21160 DSP's internal memory and external memory, external peripherals, or a host processor. DMA transfers can also occur between the ADSP-21160 DSP's internal memory and its serial ports or link ports. DMA transfers between external memory and external peripheral devices are another option. External bus packing

to 16-, 32-, 48-, or 64-bit words is automatically performed during DMA transfers.

Fourteen channels of DMA are available on the ADSP-21160 DSP—six over the link ports, four over the serial ports, and four over the processor's external port. The external port DMA channels serve for host processor, other ADSP-21160 DSPs, memory, or I/O transfers.

JTAG Port

The JTAG port on the ADSP-21160 DSP supports the IEEE standard 1149.1 Joint Test Action Group (JTAG) standard for system test. This standard defines a method for serially scanning the I/O status of each component in a system. Emulators use the JTAG port to monitor and control the DSP during emulation. Emulators using this port provide full-speed emulation with access to inspect and modify memory, registers, and processor stacks. JTAG-based emulation is non-intrusive and does not effect target system loading or timing.

Development Tools

The ADSP-21160 DSP is supported by VisualDSP++™, an easy-to-use project management environment, comprised of an Integrated Development Environment (IDDE) and Debugger. VisualDSP++ lets you manage projects from start to finish from within a single, integrated interface. Because the project development and debug environments are integrated, you can move easily between editing, building, and debugging activities.

Flexible Project Management. The VisualDSP++ IDDE provides flexible project management for the development of DSP applications. The IDDE includes access to all the activities necessary to create and debug DSP projects. You can create or modify source files or view listing or map files with the IDDE Editor. This powerful Editor is part of the IDDE and includes multiple language syntax highlighting, OLE drag and drop,

bookmarks, and standard editing operations such as undo/redo, find/replace, copy/paste/cut, and go to.

Also, the VisualDSP++ IDDE includes access to the SHARC DSP C Compiler, C Runtime Library, Assembler, Linker, Loader, Simulator, and Splitter. You specify options for these SHARC Tools through **Property Page** dialog boxes. **Property Page** dialog boxes are easy to use, and make configuring, changing, and managing your projects simple. These options control how the tools process inputs and generate outputs, and have a one-to-one correspondence to the tools' command line switches. You can define these options once, or modify them to meet changing development needs. You can also access the SHARC Tools from the operating system command line.

Greatly Reduced Debugging Time. The Debugger has an easy-to-use, common interface for all processor simulators and emulators available through Analog Devices and third parties or custom developments. The Debugger has many features that greatly reduce debugging time. You can view C source interspersed with the resulting Assembly code; profile the execution of an instruction range in a program; set simulated watch points on hardware and software registers and on program and data memory.

You can trace instruction execution and memory accesses. These features enable you to correct coding errors, identify bottlenecks, and examine DSP performance. You can use the custom register option to select any combination of registers to view in a single window. The Debugger can also generate inputs, outputs, and interrupts so you can simulate real world application conditions.

SHARC Software Development Tools. SHARC Software Development Tools, which support the SHARC DSPs, allow you to develop applications that take full advantage of the SHARC architecture, including multiprocessing, shared memory, and memory overlays. SHARC Software Development Tools include C Compiler, C Runtime Library, DSP and Math Libraries, Assembler, Linker, Loader, Simulator, and Splitter.

C Compiler and Assembler. The C Compiler generates efficient code that is optimized for both code density and execution time. The C Compiler allows you to include Assembly language statements inline. Because of this, you can program in C and still use Assembly for time-critical loops. You can also use pretested Math, DSP, and C Runtime Library routines to help shorten your time to market. The SHARC Assembly language is based on an algebraic syntax that is easy to learn, program, and debug. The add instruction, for example, is written in the same manner as the actual equation.

Linker and Loader. The Linker provides flexible system definition through Linker Description Files (`.LDF`). In a single LDF, you can define different types of executables for a single or multiprocessor system. The Linker resolves symbols over multiple executables, maximizes memory use, and easily shares common code among multiple processors. The Loader supports creation of host, link port, and PROM boot images. Along with the Linker, the Loader allows multiprocessor system configuration with smaller code and faster boot time. The Simulator is a cycle-accurate, instruction-level simulator, which enables you to simulate your application in real time.

Third-Party Extensible. The VisualDSP++ environment enables third-party companies to add value using Analog Devices' published set of Application Programming Interfaces (API). Third party products—runtime operating systems, emulators, high-level language compilers, multiprocessor hardware—can interface seamlessly with VisualDSP++ thereby simplifying the tools integration task. VisualDSP++ follows the COM API format. Two API tools, Target Wizard and API Tester, are also available for use with the API set. These tools help speed the time-to-market for vendor products. Target Wizard builds the programming shell based on API features the vendor requires. The API tester exercises the individual features independently of VisualDSP++. Third parties can use a subset of these APIs that meets their application needs. The interfaces are fully supported and backward compatible.

Further details and ordering information are available in the *VisualDSP++ Development Tools Data Sheet*. This data sheet can be requested from any Analog Devices sales office or distributor.

Differences From Previous SHARC DSPs

This section identifies differences between the ADSP-21160 DSP and previous SHARC DSPs: ADSP-21060, ADSP-21061, and ADSP-21062 processors. The ADSP-21160 DSP preserves much of the ADSP-2106x architecture, while extending performance and functionality. For background information on SHARC and the ADSP-2106x DSPs, see the *ADSP-2106x SHARC User's Manual*.

Processor Core Enhancements

Computational bandwidth on the ADSP-21160 DSP is significantly greater that on the ADSP-2106x DSPs. The increase comes from raising the operational frequency and adding another processing element: ALU, Shifter, Multiplier, and register file. The new processing element lets the DSP process multiple data streams in parallel (SIMD mode).

The program sequencer on the ADSP-21160 DSP differs from the ADSP-2106x DSP family, having several enhancements: new interrupt vector table definitions, SIMD mode stack and conditional execution model, and instruction decodes associated with new instructions. Changes to interrupts include new interrupt vectors for detecting illegal memory accesses and supporting new unshared DMA channels. Link port interrupt control has moved to a new register to support the additional DMA channels. Also, new mode stack and mode mask support has been added to improve context switch time.

Data address generators on the ADSP-21160 DSP differ from the ADSP-2106x DSPs in that DAG2 (for the PM bus) has the same addressing capability as DAG1 (for the DM bus). The DAG registers are

read/writable in pairs, moving 64-bits/cycle. Additionally, the DAGs support the new memory map and Long Word transfer capability. Circular buffering on the ADSP-21160 DSP can be quickly disabled on interrupts and restored on the return. Data “broadcast”, from one memory location to both data register files, is determined by appropriate index register usage.



Unlike previous SHARCs, the ADSP-21160 DSP has a global circular buffering enable (CBUFEN) bit. Because at reset this bit defaults to disabled, programs that use circular buffering and are being ported from previous SHARCs need to add a line of code to enable circular buffering. For more information, see “Addressing Circular Buffers” on page 4-12.

Processor Internal Bus Enhancements

The PM, DM, and IO data buses on the ADSP-21160 DSP are much wider than on the ADSP-2106x DSPs, increasing to 64 bits. Additional multiplexing and control logic on the ADSP-21160 DSP enables 16-, 32-, or 64-bit wide moves between both register files and memory.

The ADSP-21160 DSP also has the capability of broadcasting a single memory location to each of the register files in parallel. Also, the ADSP-21160 DSP permits register contents to be exchanged between the two processing elements’ register files in a single cycle.

Memory Organization Enhancements

The ADSP-21160 memory map differs from the ADSP-2106x DSPs. The system memory map on the ADSP-21160DSP supports double-word transfers each cycle, reflects extended internal memory capacity for derivative designs, and works with updated control register for SIMD support.

External Port Enhancements

The ADSP-21160 DSP's external port differs from the ADSP-2106x DSPs, greatly extending the external interface. The data bus on the ADSP-21160 DSP is 64 bits wide. The ADSP-21160 DSP has a new synchronous interface that improves local bus switching frequency. Also, burst support on the ADSP-21160 DSP improves bus usage.

Unlike previous SHARC DSPs, the ADSP-21160DSP sets the buffer hang disable (BHD) bit at reset. Because this bit prevents the processor core from detecting a buffer-related stall condition, programs that use external port, link port, or serial port I/O and are being ported from previous SHARC DSPs need to add a line of code to disable BHD. For more information, see the BHD discussion on page 6-13.

Host Interface Enhancements

The ADSP-21160's host interface differs from the ADSP-2106x DSPs in that this interface can take advantage of the 64-bit data bus width. Though the ADSP-21160 DSP supports the ADSP-2106x's asynchronous host interface protocols, the ADSP-21160 DSP also provides new synchronous interface protocols for maximum throughput.

The host/local bus deadlock resolution function on the ADSP-21160 DSP is extended to the DMA controller. The function allows the host (or bridge) logic to force the local bus to back off and allow the host to complete it's operation first.

Multiprocessor Interface Enhancements

The ADSP-21160's multiprocessor system interface supports greater throughput than the ADSP-2106x DSPs. The throughput between ADSP-21160 DSPs in a multiprocessing application increases due to shared data bus width increase to 64-bits, new shared bus transfer protocols, shared bus cycle time improvements due to synchronous interface,

and improvements in Link Port throughput. The external port supports glueless multiprocessing, with distributed arbitration for up to six ADSP-21160 DSPs.

IO Architecture Enhancements

The IO processor on the ADSP-21160 DSP provides much greater throughput than the ADSP-2106x DSPs. The Link Ports and DMA controller differ on the ADSP-21160 DSP.

DMA Controller Enhancements

The ADSP-21160's DMA controller supports 14 channels (versus 10 on the ADSP-2106x DSPs), with no channel sharing. New packing modes support the new 64-bit external/internal busing. To resolve potential deadlock scenarios, the ADSP-21106's DMA controller relinquishes the local bus in a similar fashion to the processor core when host logic asserts both $\overline{\text{HBR}}$ and $\overline{\text{SBTS}}$.

Link Port Enhancements

The ADSP-21160's Link ports provide greater throughput than the ADSP-2106x DSPs. The link port data bus width on the ADSP-21160 DSP is 8 bits wide (versus 4 bits on the ADSP-2106x DSPs). Link port clock control on the ADSP-21160 supports a wider frequency range.

Instruction Set Enhancements

ADSP-21160 DSP provides source code compatibility with the previous SHARC family members, to the application assembly source code level. All instructions, control registers, and system resources available in t the ADSP-2106x core programming model are available in ADSP-21160 DSP.

New instructions, control registers, or other facilities, required to support the new feature set of ADSP-21160 processor core are:

- Supersets of the ADSP-2106x programming model
- Reserved facilities in the ADSP-2106x programming model
- Symbol name changes from the ADSP-2106x programming model

These name changes can be managed through re-assembly using the ADSP-21160 DSP's development tools to apply the ADSP-21160 symbol definitions header file and linker description file. While these changes have no direct impact on existing core applications, system and I/O processor initialization code and control code do require modifications.

This approach simplifies porting of source code written for the ADSP-2106x DSPs to ADSP-21160 DSP. Code changes will be required to take full advantage of the new ADSP-21160 DSP features. For more information, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

For Information About Analog Products

Analog Devices is online on the internet at `http://www.analog.com`. Our Web pages provide information on the company and products, including access to technical information and documentation, product overviews, and product announcements.

You may also obtain additional information about Analog Devices and its products in any of the following ways.

- Visit our World Wide Web site at `www.analog.com`
- FAX questions or requests for information to 1(781)461-3010.
- Access the Computer Products Division File Transfer Protocol (FTP) site at `ftp ftp.analog.com` or `ftp 137.71.23.21` or `ftp://ftp.analog.com`.

For Technical or Customer Support

You can reach our Customer Support group in the following ways.

- E-mail questions to `dsp.support@analog.com` or `dsp.europe@analog.com` (European customer support)
- Contact your local ADI sales office or an authorized ADI distributor
- Send questions by mail to:

```
Analog Devices, Inc.
DSP Division
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
USA
```

What's New in This Manual

This is the third edition of the *ADSP-21160 SHARC DSP Hardware Reference*. This edition was updated to correct all open document errata.

Related Documents

For more information about Analog Devices DSPs and development products, see the following documents:

- *ADSP-21160 SHARC DSP Microcomputer Data Sheet*
- *ADSP-21160 SHARC DSP Instruction Set Reference*
- *Getting Started Guide for VisualDSP++ and ADSP-21xxx DSPs*

- *VisualDSP++ User's Guide for ADSP-21xxx DSPs*
- *VisualDSP++ C Compiler and Library Manual for ADSP-21xxx DSPs*
- *VisualDSP++ Assembler Manual for ADSP-21xxx DSPs*
- *VisualDSP++ Linker and Utilities Manual for ADSP-21xxx DSPs*

All the manuals are included in the software distribution CD-ROM. To access these manuals, use the Help Topics command in the VisualDSP++ environment's Help menu and select the Online Manuals book. From this Help topic, you can open any of the manuals, which are in Adobe Acrobat PDF format.

Conventions

The following are conventions that apply to all chapters. Note that additional conventions, which apply only to specific chapters, appear throughout this document.

Table 1-1. Notation Conventions

Example	Description
`PC`, `R1`, `PX`	Register names appear in UPPERCASE and keyword `font`
`TIMEXP`, $\overline{\texttt{RESET}}$	Pin names appear in UPPERCASE and keyword `font`; active low signals appear with an $\overline{\texttt{OVERBAR}}$.
If, Do/Until	Assembler instructions (mnemonics) appear in initial capitals
	A note, providing information of special interest or identifying a related DSP topic.

Table 1-1. Notation Conventions (Cont'd)

Example	Description
	A caution, providing information on critical design or programming issues that influence operation of the DSP.
Click Here	In the online version of this document, a cross reference acts as a hypertext link to the item being referenced. Click on blue references (Table, Figure, or section names) to jump to the location.

2 PROCESSING ELEMENTS

The DSP's Processing Elements (PEx and PEy) perform numeric processing for DSP algorithms. Each processing element contains a data register file and three computation units: an arithmetic/logic unit (ALU), a multiplier, and a shifter. Computational instructions for these elements include both fixed-point and floating-point operations, and each computational instruction can execute in a single cycle.

Overview

The computational units in a processing element handle different types of operations. The ALU performs arithmetic and logic operations on fixed-point and floating-point data. The multiplier does floating-point and fixed-point multiplication and executes fixed-point multiply/add and multiply/subtract operations. The shifter completes logical shifts, arithmetic shifts, bit manipulation, field deposit, and field extraction operations on 32-bit operands. Also, the Shifter can derive exponents.

Data flow paths through the computational units are arranged in parallel, as shown in Figure 2-1. The output of any computation unit may serve as the input of any computation unit on the next instruction cycle. Data moving in and out of the computational units goes through a 10-port register file, consisting of sixteen primary registers and sixteen alternate registers. Two ports on the register file connect to the PM and DM data buses, allowing data transfer between the computational units and memory (and anything else) connected to these buses.

The DSP's assembly language provides access to the data register files in both processing elements. The syntax lets programs move data to and from these registers and specify a computation's data format at the same time with naming conventions for the registers. For information on the data register names, see "Data Register File" on page 2-29 provides a graphical guide to the other topics in this chapter. First, a description of the MODE1 register shows how to set rounding, data format, and other modes for the processing elements. Next, an examination of each computational unit provides details on operation and a summary of computational instructions. Outside the computational units, details on register files and data buses identify how to flow data for computations. Finally, details on the DSP's advanced parallelism reveal how to take advantage of multifunction instructions and SIMD mode.

Setting Computational Modes

The MODE1 register controls the operating mode of the processing elements. Table A-2 on page A-3 lists all the bits in MODE1. The following bits in MODE1 control computational modes:

- **Floating-point data format.** Bit 16 (RND32) directs the computational units to round floating-point data to 32 bits (if 1) or round to 40 bits (if 0)
- **Rounding mode.** Bit 15 (TRUNC) directs the computational units to round results with round-to-zero (if 1) or round-to-nearest (if 0)
- **ALU saturation.** Bit 13 (ALUSAT) directs the computational units to saturate results on positive or negative fixed-point overflows (if 1) or return unsaturated results (if 0)

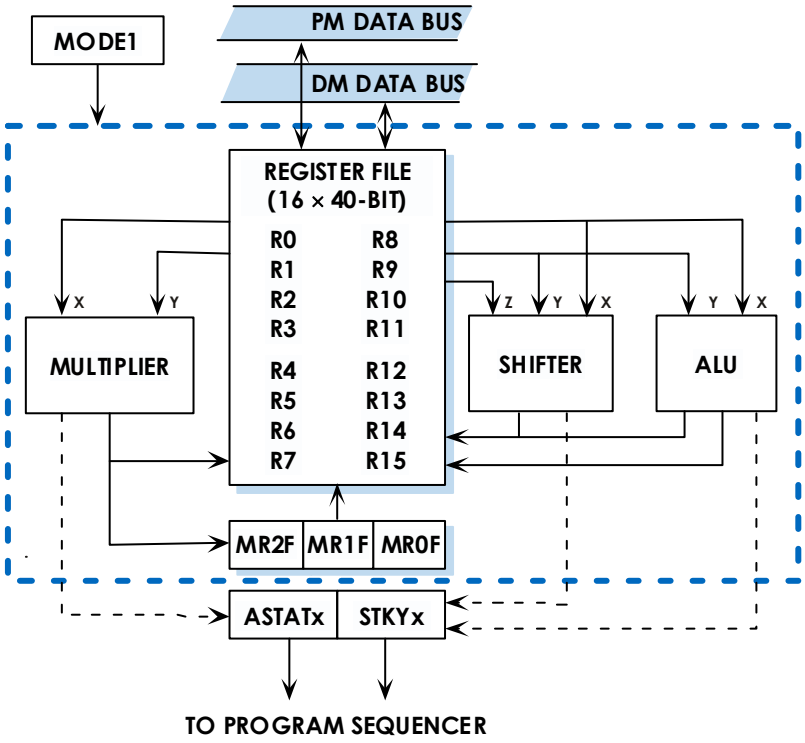


Figure 2-1. Computations Units

- **Short word sign extension.** Bit 14 (SSE) directs the computational units to sign extend short-word, 16-bit data (if 1) or zero-fill the upper 16 bits (if 0)
- **Secondary processor element (PEy).** Bit 21 (PEYEN) enables computations in PEy—SIMD mode—(if 1) or disables PEy—SISD mode—(if 0)

32-bit (Normal Word) Floating-Point Format

In the default mode of the DSP (RND32 bit=1), the multiplier and ALU support a single-precision floating-point format, which is specified in the

IEEE 754/854 standard. For more information on this standard, see "Numeric Formats". This format is IEEE 754/854 compatible for single-precision floating-point operations in all respects except that:

- The DSP does not provide inexact flags.
- NAN ("Not-A-Number") inputs generate an invalid exception and return a quiet NAN (all 1s).
- Denormal operands flush to zero when input to a computation unit and do not generate an underflow exception. Any denormal or underflow result from an arithmetic operation flushes to zero and generates an underflow exception.
- The DSP supports round to nearest and round toward zero modes, but does not support round to +Infinity and round to -Infinity.

IEEE single-precision floating-point data uses a 23-bit mantissa with an 8-bit exponent plus sign bit. In this case, the computation unit sets the eight LSBs of floating-point inputs to zeros before performing the operation. The mantissa of a result rounds to 23 bits (not including the hidden bit), and the 8 LSBs of the 40-bit result clear to zeros to form a 32-bit number, which is equivalent to the IEEE standard result.

In fixed-point to floating-point conversion, the rounding boundary is always 40 bits even if the `RND32` bit is set.

40-bit Floating-Point Format

When in extended precision mode (`RND32` bit=0), the DSP supports a 40-bit extended precision floating-point mode, which has eight additional LSBs of the mantissa and is compliant with the 754/854 standards; however, results in this format are more precise than the IEEE single-precision standard specifies. Extended-precision floating-point data uses a 31-bit mantissa with a 8-bit exponent plus sign bit.

16-bit (Short Word) Floating-Point Format

The DSP supports a 16-bit floating-point storage format and provides instructions that convert the data for 40-bit computations. The 16-bit floating-point format uses an 11-bit mantissa with a 4-bit exponent plus sign bit. The 16-bit data goes into bits 23 through 8 of a data register. Two shifter instructions, Fpack and Funpack, perform the packing and unpacking conversions between 32-bit floating-point words and 16-bit floating-point words. The Fpack instruction converts a 32-bit IEEE floating-point number in a data register into a 16-bit floating-point number. Funpack converts a 16-bit floating-point number in a data register into a 32-bit IEEE floating-point number. Each instruction executes in a single cycle.

When 16-bit data is written to bits 23 through 8 of a data register, the DSP automatically extends the data into a 32-bit integer (bits 39 through 8). If the `SSE` bit in `MODE1` is set (1), the DSP sign extends the upper 16 bits. If the `SSE` bit is cleared (0), the DSP zeros the upper 16 bits.

The 16-bit floating-point format supports gradual underflow. This method sacrifices precision for dynamic range. When packing a number that would have underflowed, the exponent clears to zero and the mantissa (including "hidden" 1) right-shifts the appropriate amount. The packed result is a denormal, which can be unpacked into a normal IEEE floating-point number.

32-Bit Fixed-Point Format

The DSP always represents fixed-point numbers in 32 bits, occupying the 32 MSBs in 40-bit data registers. Fixed-point data may be fractional or integer numbers and unsigned or twos-complement. Each computational unit has its own limitations on how these formats may be mixed for a given operation. All computational units read the upper 32 bits of data (inputs, operands) from the 40-bit registers (ignoring the 8 LSBs) and write results to the upper 32 bits (zeroing the 8 LSBs).

Rounding Mode

The TRUNC bit in the MODE1 register determines the rounding mode for all ALU operations, all floating-point multiplies, and fixed-point multiplies of fractional data. The DSP supports two modes of rounding: round-toward-zero and round-toward-nearest. The rounding modes comply with the IEEE 754 standard and have the following definitions:

- **Round-Toward-Zero** (TRUNC **bit=1**)**.** If the result before rounding is not exactly representable in the destination format, the rounded result is the number that is nearer to zero. This is equivalent to truncation.
- **Round-Toward-Nearest** (TRUNC **bit=0**)**.** If the result before rounding is not exactly representable in the destination format, the rounded result is the number that is nearer to the result before rounding. If the result before rounding is exactly halfway between two numbers in the destination format (differing by an LSB), the rounded result is the number that has an LSB equal to zero.

Statistically, rounding up occurs as often as rounding down, so there is no large sample bias. Because the maximum floating-point value is one LSB less than the value that represents Infinity, a result that is halfway between the maximum floating-point value and Infinity rounds to Infinity in this mode.

Though these rounding modes comply with standards set for floating-point data, they also apply for fixed-point multiplier operations on fractional data. The same two rounding modes are supported, but only the round-to-nearest operation is actually performed by the multiplier. Using its local result register for fixed-point operations, the multiplier rounds-to-zero by reading only the upper bits of the result and discarding the lower bits.

Using Computational Status

The multiplier and ALU each provide exception information when executing floating-point operations. Each unit updates overflow, underflow, and invalid operation flags in the processing element's arithmetic status (ASTATx and ASTATy) register and sticky status (STKYx and STKYy) register. An underflow, overflow, or invalid operation from any unit also generates a maskable interrupt. There are three ways to use floating-point exceptions from computations in program sequencing:

- **Interrupts.** Enable interrupts and use an interrupt service routine to handle the exception condition immediately. This method is appropriate if it is important to correct all exceptions as they occur.
- ASTATx **and** ASTATy **registers.** Use conditional instructions to test the exception flags in the ASTATx or ASTATy register after the instruction executes. This method permits monitoring each instruction's outcome.
- STKYx **and** STKYy **registers.** Use the Bit Tst instruction to examine exception flags in the STKY register after a series of operations. If any flags are set, some of the results are incorrect. This method is useful when exception handling is not critical.

More information on ASTAT and STKY status appears in the sections that describe the computational units. For summaries relating instructions and status bits, see Table 2-1 on page 2-11, Table 2-2 on page 2-12, Table 2-4 on page 2-19,Table 2-6 on page 2-20,and Table 2-7 on page 2-27.

Arithmetic Logic Unit (ALU)

The ALU performs arithmetic operations on fixed-point or floating-point data and logical operations on fixed-point data. ALU fixed-point instructions operate on 32-bit fixed-point operands and output 32-bit fixed-point results. ALU floating-point instructions operate on 32-bit or

40-bit floating-point operands and output 32-bit or 40-bit floating-point results. ALU instructions include:

- Floating-point addition, subtraction, add/subtract, average
- Fixed-point addition, subtraction, add/subtract, average
- Floating-point manipulation: binary log, scale, mantissa
- Fixed-point add with carry, subtract with borrow, increment, decrement
- Logical And, Or, Xor, Not
- Functions: Abs, pass, min, max, clip, compare
- Format conversion
- Reciprocal and reciprocal square root primitives

ALU Operation

ALU instructions take one or two inputs: X input and Y input. These inputs (also known as operands) can be any data registers in the register file. Most ALU operations return one result; in add/subtract operations, the ALU operation returns two results, and in compare operations, the ALU operation returns no result (only flags are updated). ALU results can be returned to any location in the register file.

The DSP transfers input operands from the register file during the first half of the cycle and transfers results to the register file during the second half of the cycle. With this arrangement, the ALU can read and write the same register file location in a single cycle. If the ALU operation is fixed-point, the inputs are treated as 32-bit fixed-point operands. The ALU transfers the upper 32 bits from the source location in the register file. For fixed-point operations, the result(s) are always 32-bit fixed-point

values. Some floating-point operations (Logb, Mant and Fix) can also yield fixed-point results.

The DSP transfers fixed-point results to the upper 32 bits of the data register and clears the lower eight bits of the register. The format of fixed-point operands and results depends on the operation. In most arithmetic operations, there is no need to distinguish between integer and fractional formats. Fixed-point inputs to operations such as scaling a floating-point value are treated as integers. For purposes of determining status such as overflow, fixed-point arithmetic operands and results are treated as twos-complement numbers.

ALU Saturation

When the ALUSAT bit is set (1) in the MODE1 register, the ALU is in saturation mode. In this mode, all positive fixed-point overflows return the maximum positive fixed-point number (0x7FFF FFFF), and all negative overflows return the maximum negative number (0x8000 0000).

When the ALUSAT bit is cleared (0) in the MODE1 register, fixed-point results that overflow are not saturated; the upper 32 bits of the result are returned unaltered.

The ALU overflow flag reflects the ALU result before saturation.

ALU Status Flags

ALU operations update seven status flags in the processing element's Arithmetic Status (ASTATx and ASTATy) register. Table A-4 on page A-9 lists all the bits in these registers. The following bits in ASTATx or ASTATy flag ALU status (a 1 indicates the condition) for the most recent ALU operation:

- **ALU result zero or floating-point underflow.** Bit 0 (AZ)
- **ALU overflow.** Bit 1 (AV)

- **ALU result negative.** Bit 2 (AN)
- **ALU fixed-point carry.** Bit 3 (AC)
- **ALU X input sign** for Abs, Mant operations. Bit 4 (AS)
- **ALU floating-point invalid operation.** Bit 5 (AI)
- **Last ALU operation was a floating-point operation.** Bit 10 (AF)
- **Compare Accumulation register results** of last 8 compare operations. Bits 31-24 (CACC)

ALU operations also update four "sticky" status flags in the processing element's Sticky status (STKYx and STKYy) register. Table A-5 on page A-13 lists all the bits in these registers. The following bits in STKYx or STKYy flag ALU status (a 1 indicates the condition). Once set, a sticky flag remains high until explicitly cleared:

- **ALU floating-point underflow.** Bit 0 (AUS)
- **ALU floating-point overflow.** Bit 1 (AVS)
- **ALU fixed-point overflow.** Bit 2 (AOS)
- **ALU floating-point invalid operation.** Bit 5 (AIS)

Flag update occurs at the end of the cycle in which the status is generated and is available on the next cycle. If a program writes the arithmetic status register or sticky status register explicitly in the same cycle that the ALU is performing an operation, the explicit write to the status register supersedes any flag update from the ALU operation.

ALU Instruction Summary

Table 2-1 and Table 2-2 list the ALU instructions and how they relate to ASTATx,y and STKYx,y flags. For more information on assembly language

syntax, see the *ADSP-21160 SHARC DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **Rn, Rx, Ry** indicate a register file location; treated as fixed-point
- **Fn, Fx, Fy** indicate a register file location; treated as floating-point
- * indicates that the flag may be set or cleared, depending on results of instruction
- ** indicates that the flag may be set (but not cleared), depending on results of instruction
- – indicates no effect

Table 2-1. Fixed-Point ALU Instruction Summary

Instruction	ASTATx,y Status Flags							STKYx,y Status Flags				
Fixed-point:	AZ	AV	AN	AC	AS	AI	AF	CACC	AUS	AVS	AOS	AIS
Rn = Rx + Ry	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx – Ry	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx + Ry + CI	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx – Ry + CI – 1	*	*	*	*	0	0	0	–	–	–	**	–
Rn = (Rx + Ry)/2	*	0	*	*	0	0	0	–	–	–	–	–
COMP(Rx, Ry)	*	0	*	0	0	0	0	*	–	–	–	–
COMPU(Rx,Ry)	*	0	*	0	0	0	0	*	--	--	--	--
Rn = Rx + CI	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx + CI – 1	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx + 1	*	*	*	*	0	0	0	–	–	–	**	–
Rn = Rx – 1	*	*	*	*	0	0	0	–	–	–	**	–
Rn = –Rx	*	*	*	*	0	0	0	–	–	–	**	–
Rn = ABS Rx	*	*	0	0	*	0	0	–	–	–	**	–

Table 2-1. Fixed-Point ALU Instruction Summary (Cont'd)

Instruction	ASTATx,y Status Flags							STKYx,y Status Flags				
Fixed-point:	**AZ**	**AV**	**AN**	**AC**	**AS**	**AI**	**AF**	**CACC**	**AUS**	**AVS**	**AOS**	**AIS**
Rn = PASS Rx	*	0	*	0	0	0	0	–	–	–	–	–
Rn = Rx AND Ry	*	0	*	0	0	0	0	–	–	–	–	–
Rn = Rx OR Ry	*	0	*	0	0	0	0	–	–	–	–	–
Rn = Rx XOR Ry	*	0	*	0	0	0	0	–	–	–	–	–
Rn = NOT Rx	*	0	*	0	0	0	0	–	–	–	–	–
Rn = MIN(Rx, Ry)	*	0	*	0	0	0	0	–	–	–	–	–
Rn = MAX(Rx, Ry)	*	0	*	0	0	0	0	–	–	–	–	–
Rn = CLIP Rx BY Ry	*	0	*	0	0	0	0	–	–	–	–	–

Table 2-2. Floating-Point ALU Instruction Summary

Instruction	ASTATx,y Status Flags							STKYx,y Status Flags				
Floating–point:	**AZ**	**AV**	**AN**	**AC**	**AS**	**AI**	**AF**	**CACC**	**AUS**	**AVS**	**AOS**	**AIS**
Fn = Fx + Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn = Fx – Fy	*	*	*	0	0	*	1	–	**	**	–	**
Fn = ABS (Fx + Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn = ABS (Fx – Fy)	*	*	0	0	0	*	1	–	**	**	–	**
Fn = (Fx + Fy)/2	*	0	*	0	0	*	1	–	**	–	–	**
COMP(Fx, Fy)	*	0	*	0	0	*	1	*	–	–	–	**
Fn = –Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = ABS Fx	*	*	0	0	*	*	1	–	–	**	–	**
Fn = PASS Fx	*	0	*	0	0	*	1	–	–	–	–	**
Fn = RND Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = SCALB Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Rn = MANT Fx	*	*	0	0	*	*	1	–	–	**	–	**

Table 2-2. Floating-Point ALU Instruction Summary (Cont'd)

Instruction	ASTATx,y Status Flags							STKYx,y Status Flags				
Floating–point:	AZ	AV	AN	AC	AS	AI	AF	CACC	AUS	AVS	AOS	AIS
Rn = LOGB Fx	*	*	*	0	0	*	1	–	–	**	–	**
Rn = FIX Fx BY Ry	*	*	*	0	0	*	1	–	**	**	–	**
Rn = FIX Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn = FLOAT Rx BY Ry	*	*	*	0	0	0	1	–	**	**	–	–
Fn = FLOAT Rx	*	0	*	0	0	0	1	–	–	–	–	–
Fn = RECIPS Fx	*	*	*	0	0	*	1	–	**	**	–	**
Fn = RSQRTS Fx	*	*	*	0	0	*	1	–	–	**	–	**
Fn = Fx COPYSIGN Fy	*	0	*	0	0	*	1	–	–	–	–	**
Fn = MIN(Fx, Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn = MAX(Fx, Fy)	*	0	*	0	0	*	1	–	–	–	–	**
Fn = CLIP Fx BY Fy	*	0	*	0	0	*	1	–	–	–	–	**

Multiply—Accumulator (Multiplier)

The multiplier performs fixed-point or floating-point multiplication and fixed-point multiply/accumulate operations. Fixed-point multiply/accumulates are available with either cumulative addition or cumulative subtraction. Multiplier floating-point instructions operate on 32-bit or 40-bit floating-point operands and output 32-bit or 40-bit floating-point results. Multiplier fixed-point instructions operate on 32-bit fixed-point data and produce 80-bit results. Inputs are treated as fractional or integer, unsigned or twos-complement. Multiplier instructions include:

- Floating-point multiplication
- Fixed-point multiplication
- Fixed-point multiply/accumulate with addition, rounding optional

- Fixed-point multiply/accumulate with subtraction, rounding optional
- Rounding result register
- Saturating result register
- Clearing result register

Multiplier Operation

The multiplier takes two inputs: X input and Y input. These inputs (also known as operands) can be any data registers in the register file. The multiplier can accumulate fixed-point results in the local Multiplier Result (MRF) registers or write results back to the register file. The results in MRF can also be rounded or saturated in separate operations. Floating-point multiplies yield floating-point results, which the multiplier always writes directly to the register file.

The multiplier transfers input operands during the first half of the cycle and transfers results during the second half of the cycle. With this arrangement, the multiplier can read and write the same register file location in a single cycle.

For fixed-point multiplies, the multiplier reads the inputs from the upper 32 bits of the data registers. Fixed-point operands may be either both in integer format or both in fractional format. The format of the result matches the format of the inputs. Each fixed-point operand may be either an unsigned or a twos-complement number. If both inputs are fractional and signed, the multiplier automatically shifts the result left one bit to remove the redundant sign bit. The register name(s) within the multiplier instruction specify input data type(s)—Fx for floating-point and Rx for fixed-point.

Multiplier (Fixed-Point) Result Register

Fixed-point operations place 80-bit results in the multiplier's foreground MRF register or background MRB register, depending on which is active. For more information on selecting the result register, see "Alternate (Secondary) Data Registers" on page 2-31.

The location of a result in the MRF register's 80-bit field depends on whether the result is in fractional or integer format, as shown in Table 2-1 on page 2-11. If the result is sent directly to a data register, the 32-bit result with the same format as the input data is transferred, using bits 63-32 for a fractional result or bits 31-0 for an integer result. The eight LSBs of the 40-bit register file location are zero-filled.

Fractional results can be rounded-to-nearest before being sent to the register file. If rounding is not specified, discarding bits 31-0 effectively truncates a fractional result (rounds to zero). For more information on rounding, see "Rounding Mode" on page 2-6.

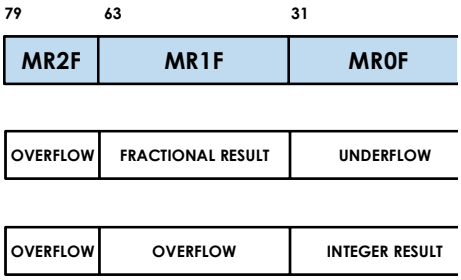


Figure 2-2. Multiplier Fixed-Point Result Placement

The MRF register is divided into MR2F, MR1F, and MR0F registers, which can be individually read from or written to the register file. Each of these registers has the same format. When data is read from MR2F, it is sign-extended to 32 bits as shown in Figure 2-3. The DSP zero fills the eight LSBs of the 40-bit register file location when data is read from MR2F, MR1F, or MR0F to the register file. When the DSP writes data into MR2F,

MR1F, or MR0F from the 32 MSBs of a register file location, the eight LSBs are ignored. Data written to MR1F is sign-extended to MR2F, repeating the MSB of MR1F in the 16 bits of MR2F. Data written to MR0F is not sign-extended.

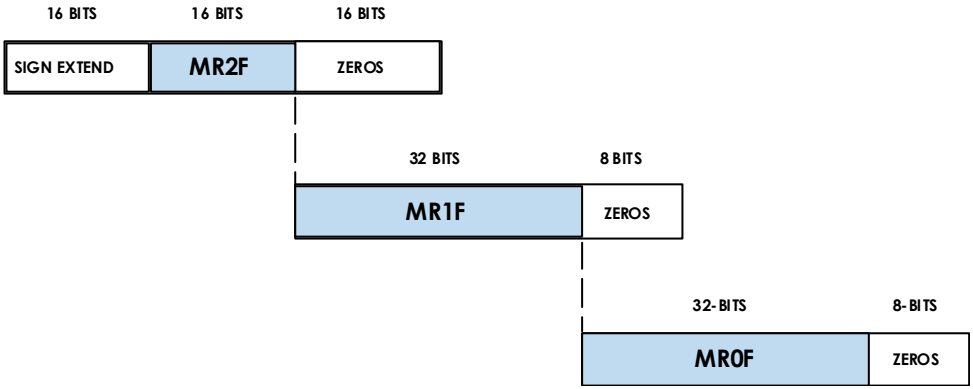


Figure 2-3. MR Transfer Formats

In addition to multiplication, fixed-point operations include accumulation, rounding and saturation of fixed-point data. There are three MRF register operations: Clear, Round, and Saturate.

The clear operation—MRF=0—resets the specified MRF register to zero. Often, it is best to perform this operation at the start of a multiply/accumulate operation to remove results left over from the previous operation.

The rounding operation—MRF=Rnd MRF—applies only to fractional results, so integer results are not effected. This operation rounds the 80-bit MRF value to nearest at bit 32; for example, the MR1F-MR0F boundary. Rounding of a fixed-point result occurs either as part of a multiply or multiply/accumulate operation or as an explicit operation on the MRF register. The rounded result in MR1F can be sent either to the register file or back to the same MRF register. To round a fractional result to zero (truncation) instead of to nearest, a program would transfer the unrounded result from MR1F, discarding the lower 32 bits in MR0F.

The saturate operation—MRF=Sat MRF—sets MRF to a maximum value if the MRF value has overflowed. Overflow occurs when the MRF value is greater than the maximum value for the data format—unsigned or twos-complement and integer or fractional—as specified in the saturate instruction. The six possible maximum values appear in Table 2-3. The result from MRF saturation can be sent either to the register file or back to the same MRF register.

Table 2-3. Fixed-Point Format Maximum Values (for Saturation)

Maximum Number	(Hexadecimal)		
	MR2F	MR1F	MR0F
2's complement fractional (positive)	0000	7FFF FFFF	FFFF FFFF
2's complement fractional (negative)	FFFF	8000 0000	0000 0000
2's complement integer (positive)	0000	0000 0000	7FFF FFFF
2's complement integer (negative)	FFFF	FFFF FFFF	8000 0000
Unsigned fractional number	0000	FFFF FFFF	FFFF FFFF
Unsigned integer number	0000	0000 0000	FFFF FFFF

Multiplier Status Flags

Multiplier operations update four status flags in the processing element's arithmetic status register (ASTATx and ASTATy). Table A-4 on page A-9 lists all the bits in these registers. The following bits in ASTATx or ASTATy flag multiplier status (a 1 indicates the condition) for the most recent multiplier operation:

- Multiplier result negative. Bit 6 (MN)
- Multiplier overflow. Bit 7 (MV)
- Multiplier underflow. Bit 8 (MU)
- Multiplier floating-point invalid operation. Bit 9 (MI)

Multiplier operations also update four “sticky” status flags in the processing element’s Sticky status (STKYx and STKYy) register. Table A-5 on page A-13 lists all the bits in these registers. The following bits in STKYx or STKYy flag multiplier status (a 1 indicates the condition). Once set, a sticky flag remains high until explicitly cleared:

- Multiplier fixed-point overflow. Bit 6 (MOS)
- Multiplier floating-point overflow. Bit 7 (MVS)
- Multiplier underflow. Bit 8 (MUS)
- Multiplier floating-point invalid operation. Bit 9 (MIS)

Flag update occurs at the end of the cycle in which the status is generated and is available on the next cycle. If a program writes the arithmetic status register or sticky register explicitly in the same cycle that the multiplier is performing an operation, the explicit write to ASTAT or STKY supersedes any flag update from the multiplier operation.

Multiplier Instruction Summary

Table 2-4 on page 2-19 and Table 2-6 on page 2-20 list the Multiplier instructions and how they relate to ASTATx,y and STKYx,y flags. For more information on assembly language syntax, see the *ADSP-21160 SHARC DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **Rn, Rx, Ry** indicate any register file location; treated as fixed-point
- **Fn, Fx, Fy** indicate any register file location; treated as floating-point
- * indicates the flag may be set or cleared, depending on results of instruction

- ** indicates the flag may be set (but not cleared), depending on results of instruction
- – indicates no effect
- The **Input Mods** column indicates the types of optional modifiers that you can apply to the instructions inputs. For a list of modifiers, see Table 2-5.

Table 2-4. Fixed-Point Multiplier Instruction Summary

Instruction	Input Mods	ASTATx,y Flags				STKYx,y Flags			
Fixed-Point: For Input Mods, see Table 2-5		MU	MN	MV	MI	MUS	MOS	MVS	MIS
Rn = Rx * Ry	1	*	*	*	0	–	**	–	–
MRF = Rx * Ry	1	*	*	*	0	–	**	–	–
MRB = Rx * Ry	1	*	*	*	0	–	**	–	–
Rn = MRF + Rx * Ry	1	*	*	*	0	–	**	–	–
Rn = MRB + Rx * Ry	1	*	*	*	0	–	**	–	–
MRF = MRF + Rx * Ry	1	*	*	*	0	–	**	–	–
MRB = MRB + Rx * Ry	1	*	*	*	0	–	**	–	–
Rn = MRF – Rx * Ry	1	*	*	*	0	–	**	–	–
Rn = MRB – Rx * Ry	1	*	*	*	0	–	**	–	–
MRF = MRF – Rx * Ry	1	*	*	*	0	–	**	–	–
MRB = MRB – Rx * Ry	1	*	*	*	0	–	**	–	–
Rn = SAT MRF	2	*	*	*	0	–	**	–	–
Rn = SAT MRB	2	*	*	*	0	–	**	–	–
MRF = SAT MRF	2	*	*	*	0	–	**	–	–
MRB = SAT MRB	2	*	*	*	0	–	**	–	–
Rn = RND MRF	3	*	*	*	0	–	**	–	–
Rn = RND MRB	3	*	*	*	0	–	**	–	–
MRF = RND MRF	3	*	*	*	0	–	**	–	–

Table 2-4. Fixed-Point Multiplier Instruction Summary (Cont'd)

Instruction	Input Mods	ASTATx,y Flags				STKYx,y Flags			
Fixed-Point: For Input Mods, see Table 2-5		MU	MN	MV	MI	MUS	MOS	MVS	MIS
MRB = RND MRB	3	*	*	*	0	–	**	–	–
MRF= 0	–	–	–	–	–	–	–	–	–
MRB= 0	–	–	–	–	–	–	–	–	–
MRxF = Rn	–	–	–	–	–	–	–	–	–
MRxB = Rn	–	–	–	–	–	–	–	–	–
Rn = MRxF	–	–	–	–	–	–	–	–	–
Rn = MRxB	–	–	–	–	–	–	–	–	–

Table 2-5. Input Modifiers for Fixed-Point Multiplier Instruction

Input Mods from Table 2-2	Input Mods—Options For Fixed-point Multiplier Instructions
	Note the meaning of the following symbols in this table: **SSigned input** **UUnsigned input** **IInteger input(s)** **FFractional input(s)** **FRFractional inputs, Rounded output** **Note that (SF) is the default format for 1-input operations, and (SSF) is the default format for 2-input operations**
1	(SSF), (SSI), (SSFR), (SUF), (SUI), (SUFR), (USF), (USI), (USFR), (UUF), (UUI), or (UUFR)
2	(SF), (SI), (UF), or (UI)
3	(SF) or (UF)

Table 2-6. Floating-Point Multiplier Instruction Summary

Instruction	ASTATx,y Flags				STKYx,y Flags			
Floating-Point:	MU	MN	MV	MI	MUS	MOS	MVS	MIS
Fn = Fx * Fy	*	*	*	*	**	–	**	**

Barrel-Shifter (Shifter)

The shifter performs bit-wise operations on 32-bit fixed-point operands. Shifter operations include:

- Shifts and rotates from off-scale left to off-scale right
- Bit manipulation operations, including bit set, clear, toggle, and test
- Bit field manipulation operations, including extract and deposit
- Fixed-point/floating-point conversion operations, including exponent extract, number of leading 1s or 0s

Shifter Operation

The shifter takes from one to three inputs: X-input, Y-input, and Z-input. The inputs (also known as operands) can be any register in the register file. Within a shifter instruction, the inputs serve as follows:

- The X-input provides data that is operated on
- The Y-input specifies shift magnitudes, bit field lengths or bit positions
- The Z-input provides data that is operated on and updated

In the following example, Rx is the X-input, Ry is the Y-input, and Rn is the Z-input. The shifter returns one output (Rn) to the register file.

```
Rn = Rn OR LSHIFT Rx BY Ry;
```

As shown in Figure 2-4, the shifter fetches input operands from the upper 32 bits of a register file location (bits 39-8) or from an immediate value in the instruction. The shifter transfers operands during the first half of the cycle and transfers the result to the upper 32 bits of a register (with the eight LSBs zero-filled) during the second half of the cycle. With this

arrangement, the shifter can read and write the same register file location in a single cycle.

The X-input and Z-input are always 32-bit fixed-point values. The Y-input is a 32-bit fixed-point value or an 8-bit field (shf8), positioned in the register file. These inputs appear in Figure 2-4.

Some shifter operations produce 8-bit or 6-bit results. As shown in Figure 2-5, the shifter places these results in either the shf8 field or the bit6 field and sign-extends the results to 32 bits. The shifter always returns a 32-bit result.

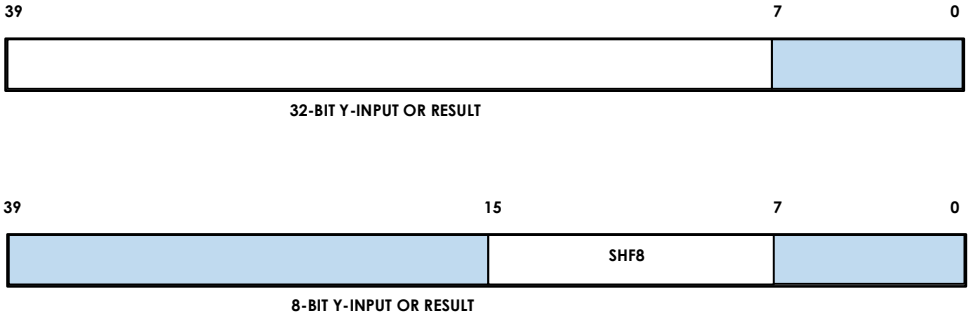


Figure 2-4. Register File Fields for Shifter Instructions

The shifter supports bit field deposit and bit field extract instructions for manipulating groups of bits within an input. The Y-input for bit field instructions specifies two 6-bit values: bit6 and len6, which are positioned in the Ry register as shown in Figure 2-5. The shifter interprets bit6 and len6 as positive integers. Bit6 is the starting bit position for the deposit or

extract, and len6 is the bit field length, which specifies how many bits are deposited or extracted.

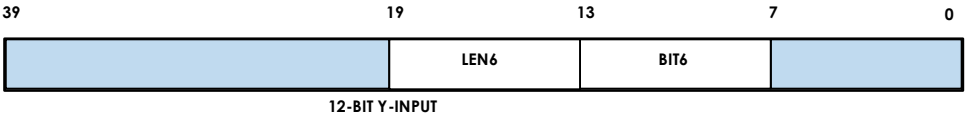


Figure 2-5. Register File Fields for FDEP and FEXT Instructions

Field deposit (Fdep) instructions take a group of bits from the input register (starting at the LSB of the 32-bit integer field) and deposit the bits as directed anywhere within the result register. The bit6 value specifies the starting bit position for the deposit. Figure 2-6 shows bit placement for the following field deposit instruction:

```
R0 = FDEP R1 BY R2;
```

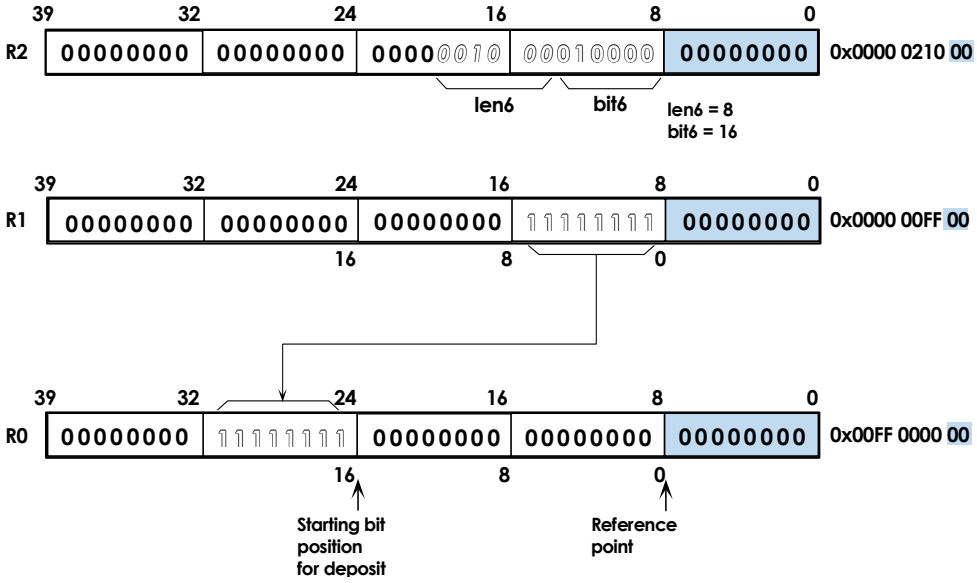


Figure 2-6. Bit Field Deposit Example

Figure 2-7 shows how the inputs, bit6 and len6, work in an field deposit instruction (Rn=Fdep Rx By Ry). Field extract (`Fext`) instructions extract a group of bits as directed from anywhere within the input register and place them in the result register (aligned with the LSB of the 32-bit integer field). The bit6 value specifies the starting bit position for the extract.

Figure 2-8 shows bit placement for the following field extract instruction:

```
R3 = FEXT R4 BY R5;
```

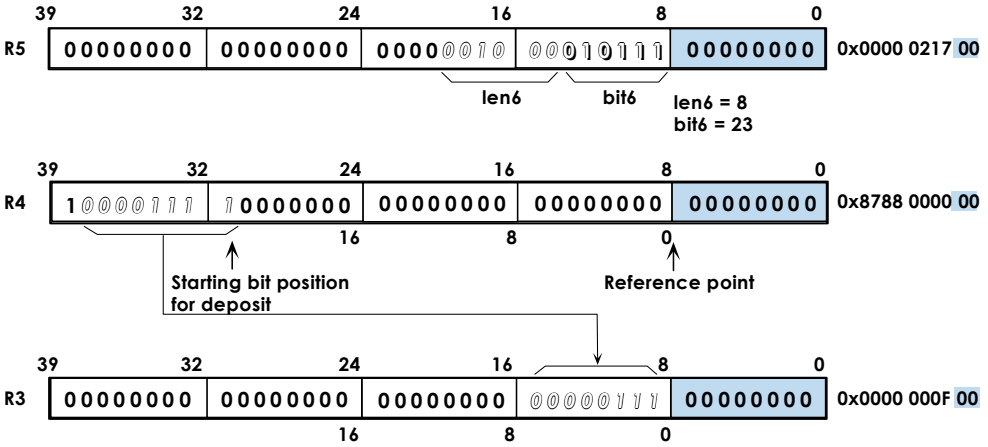


Figure 2-7. Bit Field Extract Example

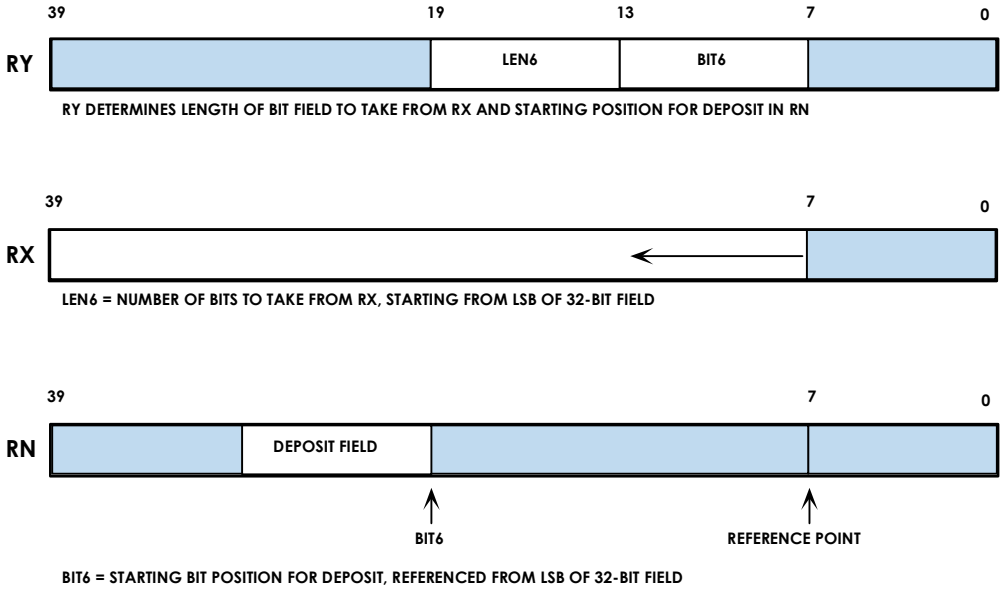


Figure 2-8. Bit Field Deposit Instruction

Shifter Status Flags

Shifter operations update three status flags in the processing element's arithmetic status register (ASTATx and ASTATy). Table A-4 on page A-9 lists all the bits in these registers. The following bits in ASTATx or ASTATy flag shifter status (a 1 indicates the condition) for the most recent ALU operation:

- Shifter overflow of bits to left of MSB. Bit 11 (SV)
- Shifter result zero. Bit 12 (SZ)
- Shifter input sign for exponent extract only. Bit 13 (SS)

Flag update occurs at the end of the cycle in which the status is generated and is available on the next cycle. If a program writes the arithmetic status register explicitly in the same cycle that the shifter is performing an opera-

tion, the explicit write to ASTAT supersedes any flag update caused by the shift operation.

Shifter Instruction Summary

Table 2-7 on page 2-27 lists the Shifter instructions and how they relate to ASTATx,y flags. For more information on assembly language syntax, see the *ADSP-21160 SHARC DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **Rn, Rx, Ry** indicate any register file location; bit fields used depend on instruction
- **Fn, Fx** indicate any register file location; floating-point word
- * indicates the flag may set or cleared, depending on data

Table 2-7. Shifter Instruction Summary

Instruction	ASTATx,y Flags		
	SZ	SV	SS
Rn = LSHIFT Rx BY Ry	*	*	0
Rn = LSHIFT Rx BY <data8>	*	*	0
Rn = Rn OR LSHIFT Rx BY Ry	*	*	0
Rn = Rn OR LSHIFT Rx BY <data8>	*	*	0
Rn = ASHIFT Rx BY Ry	*	*	0
Rn = ASHIFT Rx BY<data8>	*	*	0
Rn = Rn OR ASHIFT Rx BY Ry	*	*	0
Rn = Rn OR ASHIFT Rx BY <data8>	*	*	0
Rn = ROT Rx BY Ry	*	0	0
Rn = ROT Rx BY <data8>	*	0	0
Rn = BCLR Rx BY Ry	*	*	0
Rn = BCLR Rx BY <data8>	*	*	0
Rn = BSET Rx BY Ry	*	*	0

Table 2-7. Shifter Instruction Summary (Cont'd)

Instruction	ASTATx,y Flags		
	SZ	SV	SS
Rn = BSET Rx BY <data8>	*	*	0
Rn = BTGL Rx BY Ry	*	*	0
Rn = BTGL Rx BY <data8>	*	*	0
BTST Rx BY Ry	*	*	0
BTST Rx BY <data8>	*	*	0
Rn = FDEP Rx BY Ry	*	*	0
Rn = FDEP Rx BY <bit6>:<len6>	*	*	0
Rn = Rn OR FDEP Rx BY Ry	*	*	0
Rn = Rn OR FDEP Rx BY <bit6>:<len6>	*	*	0
Rn = FDEP Rx BY Ry (SE)	*	*	0
Rn = FDEP Rx BY <bit6>:<len6> (SE)	*	*	0
Rn = Rn OR FDEP Rx BY Ry (SE)	*	*	0
Rn = Rn OR FDEP Rx BY <bit6>:<len6> (SE)	*	*	0
Rn = FEXT Rx BY Ry	*	*	0
Rn = FEXT Rx BY <bit6>:<len6>	*	*	0
Rn = FEXT Rx BY Ry (SE)	*	*	0
Rn = FEXT Rx BY <bit6>:<len6> (SE)	*	*	0
Rn = EXP Rx (EX)	*	0	*
Rn = EXP Rx	*	0	*
Rn = LEFTZ Rx	*	*	0
Rn = LEFTO Rx	*	*	0
Rn = FPACK Fx	0	*	0
Fn = FUNPACK Rx	0	0	0

Data Register File

Each of the DSP's processing elements has a data register file: a set of data registers that transfer data between the data buses and the computation units. These registers also provide local storage for operands and results.

The two register files each consist of 16 primary registers and 16 alternate (secondary) registers. All of the data registers are 40 bits wide. Within these registers, 32-bit data is always left-justified. If an operation specifies a 32-bit data transfer to these 40-bit registers, the eight LSBs are ignored on register reads, and the eight LSBs are cleared to zeros on writes.

Program memory data accesses and data memory accesses to/from the register file(s) occur on the PM data bus and DM data bus, respectively. One PM data bus access for each processing element and/or one DM data bus access for each processing element can occur in one cycle. Transfers between the register files and the DM or PM data buses can move up to 64-bits of valid data on each bus.

If an operation specifies the same register file location as both an input and output, the read occurs in the first half of the cycle and the write in the second half. With this arrangement, the DSP uses the old data as the operand, before updating the location with the new result data. If writes to the same location take place in the same cycle, only the write with higher precedence actually occurs. The DSP determines precedence for the write operation from the source of the data; from highest to lowest, the precedence is:

1. Data memory or universal register
2. Program memory
3. PEx ALU
4. PEy ALU
5. PEx Multiplier

6. PEy Multiplier
7. PEx Shifter
8. PEy Shifter

The data register file in Figure 2-1 on page 2-3 lists register names of R0 through R15 within PEx's register file. When a program refers to these registers as R0 through R15, the computational units treat the registers' contents as fixed-point data. To perform floating point computations, refer to these registers as F0 through F15. For example, the following instructions refer to the same registers, but direct the computational units to perform different operations:

```
F0=F1 * F2; floating-point multiply
R0=R1 * R2; fixed-point multiply
```

The F and R prefixes on register names do not effect the 32-bit or 40-bit data transfer; the naming convention only determines how the ALU, multiplier, and shifter treat the data.



To maintain compatibility with code written for previous SHARC DSPs, the assembly syntax accommodates references to PEx data registers and PEy data registers.

Code may only refer to the PEy data registers (S0 through S15) for data move instructions. The rules for using register names are as follows:

- R0 through R15 and F0 through F15 always refer to PEx registers for data move and computational instructions, whether the DSP is in SISD or SIMD mode
- R0 through R15 and F0 through F15 refer to both PEx and PEy register for computational instructions in SIMD mode
- S0 through S15 always refer to PEy registers for data move instructions, whether the DSP is in SISD or SIMD mode

For more information on SISD and SIMD computational operations, see "Secondary Processing Element (PEy)" on page 2-36. For more information on ADSP-21160 assembly language, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Alternate (Secondary) Data Registers

Each register file has an alternate register set. To facilitate fast context switching, the DSP includes alternate register sets for data, results, and data address generator registers. Bits in the MODE1 register control when alternate registers become accessible. While inaccessible, the contents of alternate registers are not effected by DSP operations. Note that there is a one cycle latency between writing to MODE1 and being able to access an alternate register set. The alternate register sets for data and results are described in this section. For more information on alternate data address generator registers, see "Alternate (Secondary) Data Registers" on page 2-31.

Bits in the MODE1 register can activate independent-alternate-data-register sets: the lower half (R0-R7 and S0-S7) and the upper half (R8-R15 and S8-S15). To share data between contexts, a program places the data to be shared in one half of either the current processing element's register file or the opposite processing element's register file and activates the alternate register set of the other half. For information on how to activate alternate data registers, see the description on page 2-31.

Each multiplier has a primary or foreground (MRF) register and alternate or background (MRB) results register. A bit in the MODE1 register selects which result register receives the result from the multiplier operation, swapping which register is the current MRF or MRB. This swapping facilitates context switching. Unlike other registers that have alternates, both MRF and MRB are accessible at the same time. All fixed-point multiplies can accumulate results in either MRF or MRB, without regard to the state of the MODE1 register. With this arrangement, code can use the result registers as primary

and alternate accumulators, or code can use these registers as two parallel accumulators. This feature facilitates complex math.

The MODE1 register controls the access to alternate registers. Table A-2 on page A-3 lists all the bits in MODE1. The following bits in MODE1 control alternate registers (a 1 enables the alternate set):

- Secondary registers for computation unit results. Bit 2 (SRCU)
- Secondary registers for hi register file, R8-R15 and S8-15. Bit 7 (SRRFH)
- Secondary registers for lo register file, R0-R7 and S0-S7. Bit 10 (SRRFL)

The following example demonstrates how code should handle the one cycle of latency from the instruction setting the bit in MODE1 to when the alternate registers may be accessed.

```
BIT SET MODE1 SRRFL;/* activate alternate reg. file */
NOP;/* wait for access to alternates */
R0=7;
```

Multifunction Computations

Using the many parallel data paths within its computational units, the DSP supports multiple-parallel (multifunction) computations. These instructions complete in a single cycle, and they combine parallel operation of the multiplier and the ALU or dual ALU functions. The multiple operations perform the same as if they were in corresponding single-function computations. Multifunction computations also handle flags in the same way as the single-function computations, except that in the dual add/subtract computation the ALU flags from the two operations are Or'ed together.

To work with the available data paths, the computation units constrain which data registers may hold the four input operands for multifunction computations. These constraints limit which registers may hold the X-input and Y-input for the ALU and multiplier.

Figure 2-9 shows a computational unit and indicates which registers may serve as X-inputs and Y-inputs for the ALU and multiplier.

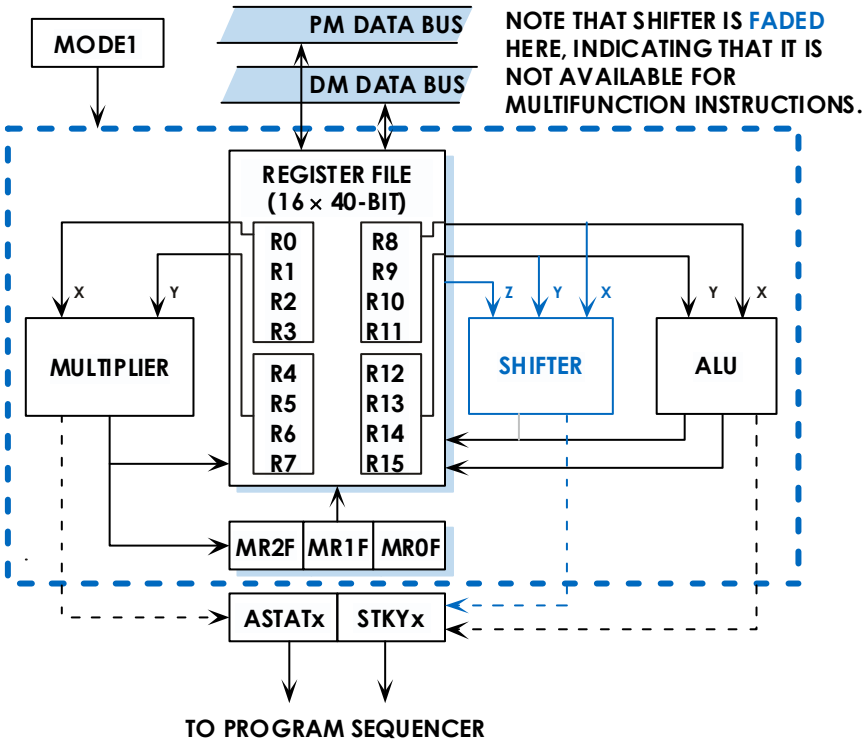


Figure 2-9. Input Registers for Multifunction Computations (ALU and Multiplier)

For example, the X-input to the ALU can only be `R8`, `R9`, `R10` or `R11`. Note that the shifter is gray in Figure 2-9 to indicate that there are no shifter multifunction operations.

Table 2-8, Table 2-9, Table 2-10, and Table 2-11 list the multifunction computations. For more information on assembly language syntax, see the *ADSP-21160 SHARC DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **Rm, Ra, Rs, Rx, Ry** indicate any register file location; fixed-point
- **Fm, Fa, Fs, Fx, Fy** indicate any register file location; floating-point
- **R3-0** indicates data file registers `R3`, `R2`, `R1`, or `R0`, and **F3-0** indicates data file registers `F3`, `F2`, `F1`, or `F0`
- **R7-4** indicates data file registers `R7`, `R6`, `R5`, or `R4`, and **F7-4** indicates data file registers `F7`, `F6`, `F5`, or `F4`
- **R11-8** indicates data file registers `R11`, `R10`, `R9`, or `R8`, and **F11-8** indicates data file registers `F11`, `F10`, `F9`, or `F8`
- **R15-12** indicates data file registers `R15`, `R14`, `R13`, or `R12`, and **F15-12** indicates data file registers `F15`, `F14`, `F13`, or `F12`
- **SSFR** indicates the X-input is signed, Y-input is signed, use Fractional inputs, and Rounded-to-nearest output
- **SSF** indicates the X-input is signed, Y-input is signed, use Fractional input

Table 2-8. Dual Add And Subtract

Ra = Rx + Ry, Rs = Rx – Ry
Fa = Fx + Fy, Fs = Fx – Fy

Table 2-9. Fixed-Point Multiply and Add, Subtract, or Average

(Any combination of left and right column)		
Rm=R3-0 * R7-4 (SSFR),		Ra=R11-8 + R15-12
MRF=MRF + R3-0 * R7-4 (SSF),		Ra=R11-8 – R15-12
Rm=MRF + R3-0 * R7-4 (SSFR),		Ra=(R11-8 + R15-12)/2
MRF=MRF – R3-0 * R7-4 (SSF),		
Rm=MRF – R3-0 * R7-4 (SSFR),		

Table 2-10. Floating-Point Multiply And ALU Operation

Fm=F3-0 * F7-4, Fa=F11-8 + F15-12
Fm=F3-0 * F7-4, Fa=F11-8 – F15-12
Fm=F3-0 * F7-4, Fa=FLOAT R11-8 by R15-12
Fm=F3-0 * F7-4, Ra=FIX F11-8 by R15-12
Fm=F3-0 * F7-4, Fa=(F11-8 + F15-12)/2
Fm=F3-0 * F7-4, Fa=ABS F11-8
Fm=F3-0 * F7-4, Fa=MAX (F11-8, F15-12)
Fm=F3-0 * F7-4, Fa=MIN (F11-8, F15-12)

Table 2-11. Multiply With Dual Add and Subtract

Rm = R3-0 * R7-4 (SSFR), Ra = R11-8 + R15-12, Rs = R11-8 – R15-12
Fm = F3-0 * F7-4, Fa = F11-8 + F15-12, Fs = F11-8 – F15-12

Another type of multifunction operation is also available on the DSP, combining transfers between the results and data registers and transfers between memory and data registers. Like other multifunction instructions, these parallel operations complete in a single cycle. For example, the DSP can perform the following multiply and parallel read of data memory:

```
MRF=MRF-R5*R0, R6=DM(I1,M2);
```

Or, the DSP can perform the following result register transfer and parallel read:

```
R5=MR1F, R6=DM(I1,M2);
```

Secondary Processing Element (PEy)

The ADSP-21160 DSP contains two sets of computation units and associated register files. As shown in Figure 2-10 on page 2-35, these two Processing Elements (PEx and PEy) support Single Instruction, Multiple Data (SIMD) operation.

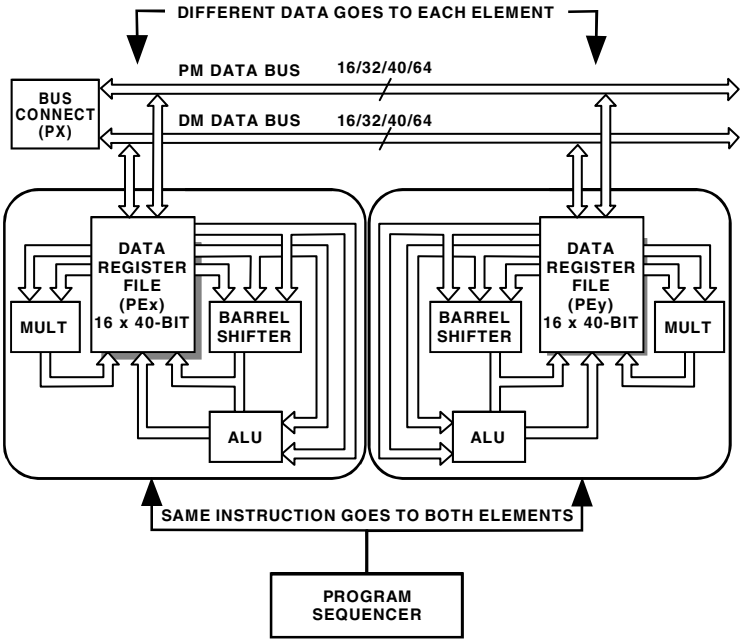


Figure 2-10. Block Diagram Showing Secondary Execution

The MODE1 register controls the operating mode of the processing elements. Table A-2 on page A-3 lists all the bits in MODE1.

The PEYEN bit (bit 21) in the MODE1 register enables or disables the PEy processing element. When PEYEN is cleared (0), the ADSP-21160 DSP operates in Single-Instruction-Single-Data (SISD) mode, using only PEx; this is the mode in which ADSP-2106x DSPs operate. When the PEYEN bit is set (1), the ADSP-21160 DSP operates in SIMD mode, using the PEx and PEy processing elements. There is a one cycle delay after PEYEN is set or cleared, before the change to or from SIMD mode takes effect.

To support SIMD, the DSP performs the following parallel operations:

- Dispatches a single instruction to both processing element's computation units
- Loads two sets of data from memory, one for each processing element
- Executes the same instruction simultaneously in both processing elements
- Stores data results from the dual executions to memory

Using the information here and in the *ADSP-21160 SHARC DSP Instruction Set Reference*, it is possible through SIMD mode's parallelism to double performance over similar algorithms running in SISD (ADSP-2106x DSP compatible) mode.

The two processing elements are symmetrical, each containing the following functional blocks:

- ALU
- Multiplier primary and alternate result registers
- Shifter
- Data register file and alternate register file

Dual Compute Units Sets

The computation units (ALU, Multiplier, and Shifter) in PEx and PEy are identical. The data bus connections for the dual computation units permit asymmetric data moves to, from, and between the two processing elements. Identical instructions execute on the PEx and PEy computational units; the difference is the data. The data registers for PEy operations are identified (implicitly) from the PEx registers in the instruction.

This implicit relation between PEx and PEy data registers corresponds to complementary register pairs in Table 2-12. Any universal registers that do not appear in Table 2-12 have the same identities in both PEx and PEy. When a computation in SIMD mode refers to a register in the PEx column, the corresponding computation in PEy refers to the complimentary register in the PEy column.

Table 2-12. SIMD Mode Complementary Register Pairs

PEx	PEy
R0	S0
R1	S1
R2	S2
R3	S3
R4	S4
R5	S5
R6	S6
R7	S7
R8	S8
R9	S9
R10	S10
R11	S11
R12	S12
R13	S13
R14	S14
R15	S15
USTAT1	USTAT2
USTAT3	USTAT4
ASTATx	ASTATy
STKYx	STKYy
PX1	PX2

Dual Register Files

The two 16 entry data register files (one in each PE) and their operand and result busing and porting are identical. The same is true for each 16 entry alternate register files. The transfer direction, source and destination registers, and data bus usage depend on the following conditions:

- Computational mode:
  - Is PEy enabled (PEYEN bit=1 in MODE1 register)?
  - Is the data register file in PEx (R0-R15, F0-F15) or PEy (S0-S15)?
  - Is the instruction a data register swap between processing elements?
- Data addressing mode:
- What is the state of the Internal Memory Data Width (IMDW) bits in the System Configuration (SYSCON) register?
- Is Broadcast write enabled (BDCST1,9 bits in MODE1 register)?
- What is the type of address (long, normal, or short word)?
- Is long-word override (LW) specified in the instruction?
- What are the states of instruction fields for DAG1 or DAG2?
- Program sequencing (conditional logic):
- What is the outcome of the instruction's condition comparison on each processing element?

For information on SIMD issues that relate to computational modes, see "SIMD (Computational) Operations" on page 2-40. For information on SIMD issues relating to data addressing, see "SIMD Mode and Sequenc-

ing" on page 3-57. For information on SIMD issues relating to program sequencing, see "Addressing in SISD and SIMD Modes" on page 4-19.

Dual Alternate Registers

Both register files consist of a primary set of 16 by 40-bit registers and an alternate set of 16 by 40-bit registers. Context switching between the two sets of registers occur in parallel between the two processing elements. For more information, see "Alternate (Secondary) Data Registers" on page 2-31.

SIMD (Computational) Operations

In SIMD mode, the dual processing elements execute the same instruction, but operate on different data. To support SIMD operation, the elements support a variety of dual data move features.

The DSP supports unidirectional and bidirectional register-to-register transfers with the conditional compute and move instruction. All four combinations of inter-register file and intra-register file transfers (PEx <-> PEx, PEx <-> PEy, PEy <-> PEx, and PEy <-> PEy) are possible in both SISD (unidirectional) and SIMD (bidirectional) modes.

In SISD mode (PEYEN bit=0), the register-to-register transfers are unidirectional, meaning that an operation performed on one processing element is not duplicated on the other processing element. The SISD transfer uses a source register and a destination register, and either register can be in either element's data register file. For a summary of unidirectional transfers, see the upper half of Table 2-12 on page 2-38. Note that in SISD mode a condition for an instruction only tests in the PEx element and applies to the entire instruction.

In SIMD mode (PEYEN bit=1), the register-to-register transfers are bidirectional, meaning that an operation performed on one element is duplicated in parallel on the other element. The instruction uses two source registers

(one from each element's register file) and two destination registers (one from each element's register file).

For a summary of bidirectional transfers, see the lower half of Table 2-12 on page 2-38. Note that in SIMD mode a conditional for an instruction test in both the PEx and PEy elements, dividing control of the explicit and implicit transfers as detailed in Table 2-12 on page 2-38.

Bidirectional register-to-register transfers in SIMD mode are allowed between a data register and DAG, control, or status registers. When the DAG, control, or status register is a source of the transfer, the destination can be a data register. This SIMD transfer duplicates the contents of the source register in a data register in both processing elements.



Careful programming is required when a DAG, control, or status register is a destination of a transfer from a data register. If the destination register has a complement (for example ASTATx and ASTATy), the SIMD transfer moves the contents of the explicit data register into the explicit destination and moves the contents of the implicit data register into the implicit destination (the complement). If the destination register has no complement (for example, I0), only the explicit transfer occurs.

Even if the code uses a conditional operation to select whether the transfer occurs, only the explicit transfer can take place if the destination register has no complement.

In the case where a DAG, control, or status register is both source and destination, the data move operation executes the same as if SIMD mode were disabled.

In both SISD and SIMD modes, the DSP supports bidirectional register-to-register swaps. The swap always occurs between one register in each processing element's data register file.

Registers swaps use the special swap operator, <->. A register-to-register swap occurs when registers in different processing elements exchange val-

ues; for example `R0 <-> S1`. Only single, 40-bit register to register swaps are supported—no double register operations.

When they are unconditional, register-to-register swaps operate the same in SISD mode and SIMD mode. If a condition is added to the instruction in SISD mode, the condition tests only in the PEx element and controls the entire operation. If a condition is added in SIMD mode, the condition tests in both the PEx and PEy elements and controls the halves of the operation as detailed in Table 2-12 on page 2-38.

Table 2-13. Register-to-Register Move Summary (SISD versus SIMD)

Mode	Instruction	Explicit Transfer	Implicit Transfer
SISD[1]	IF condition compute, Rx = Ry;	Rx loaded from Ry	None
	IF condition compute, Rx = Sy;	Rx loaded from Sy	None
	IF condition compute, Sx = Ry;	Sx loaded from Ry	None
	IF condition compute, Sx = Sy;	Sx loaded from Sy	None
	IF condition compute, Rx <-> Sy;	Rx swaps to Sy Sy swaps to Rx	None
SIMD[2]	IF condition compute, Rx = Ry;	Rx loaded from Ry	Sx loaded from Sy
	IF condition compute, Rx = Sy;	Rx loaded from Sy	Sx loaded from Ry
	IF condition compute, Sx = Ry;	Sx loaded from Ry	Rx loaded from Sy
	IF condition compute, Sx = Sy;	Sx loaded from Sy	Rx loaded from Ry
	IF condition compute, Rx <-> Sy;[3]	Sy swaps to Rx	Rx swaps to Sy

1 In SISD mode, the conditional applies only to the entire operation and is only tested against PEx's flags. When the condition tests true, the entire operation occurs.

2 In SIMD mode, the conditional applies separately to the explicit and implicit transfers. Where the condition tests true (PEx for the explicit and PEy for the implicit), the operation occurs in that processing element.

3 Register to register transfers (R0=S0) and register swaps (R0<->S0) do not cause a PMD bus conflict. These operations use only the DMD bus and a hidden 16-bit bus to do the two register moves.

SIMD conditional instructions with the same destination registers do not produce predictable transfers. For example, the instruction `IF EQ R4 = R14 - R15, S4 = R6`; may not work as expected. This kind of usage is prohibited, as it is not logical to use it this way.

SIMD and Status Flags

When the DSP is in SIMD mode (`PEYEN` bit=1), computations on both processing elements generate status flags, producing a logical Or'ing of the exception status test on each processing element. If one of the four fixed-point or floating-point exceptions is enabled, an exception condition on either or both processing elements generates an exception interrupt. Interrupt service routines must determine which of the processing elements encountered the exception. Note that returning from a floating point interrupt does not automatically clear the `STKY` state. Code must clear the `STKY` bits in both processing element's sticky status (`STKYx` and `STKYy`) registers as part of the exception service routine. For more information, see For more information, see "Interrupts and Sequencing" on page 3-33.

3 PROGRAM SEQUENCER

The DSP's program sequencer implements program flow, constantly providing the address of the next instruction to be executed by other parts of the DSP.

Overview

Program flow in the DSP is mostly linear with the processor executing program instructions sequentially. This linear flow varies occasionally when the program uses non-sequential program structures, such as those illustrated in Figure 3-1. Non-sequential structures direct the DSP to execute an instruction that is not at the next sequential address, following the current instruction. These structures include:

Loops. One sequence of instructions executes several times with zero overhead.

- **Subroutines.** The processor temporarily interrupts sequential flow to execute instructions from another part of program memory.
- **Jumps.** Program flow transfers permanently to another part of program memory.

- **Interrupts.** Subroutines in which a runtime event (not an instruction) triggers the execution of the routine.
- **Idle.** An instruction that causes the processor to cease operations, holding its current state until an interrupt occurs. Then, the processor services the interrupt and continues normal execution.

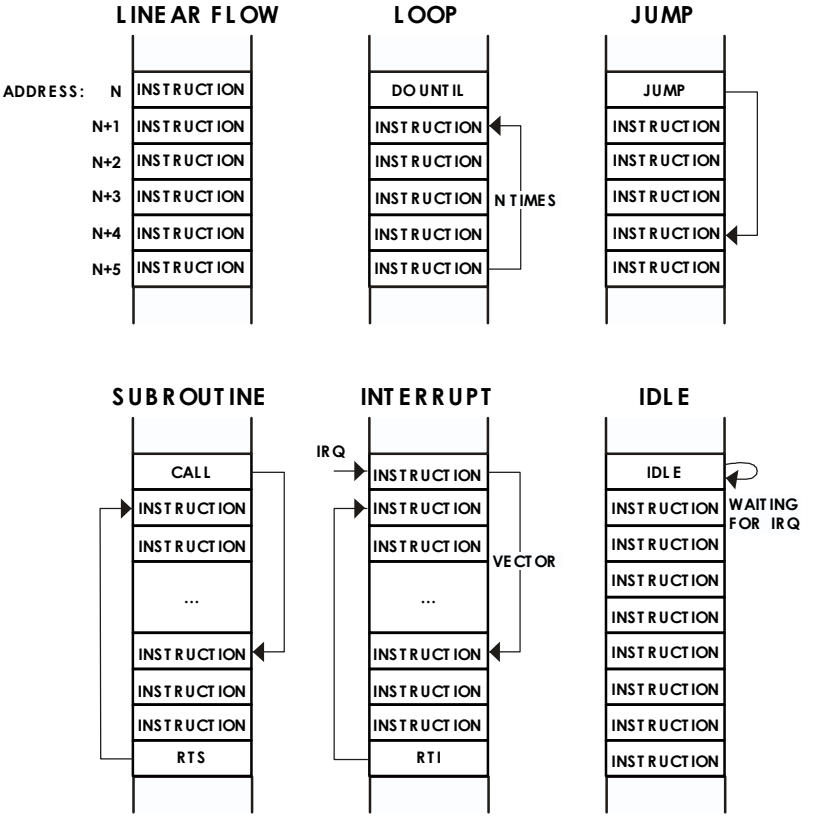


Figure 3-1. Program Flow Variations

The sequencer manages execution of these program structures by selecting the address of the next instruction to execute. As part of its process, the sequencer handles the following tasks.

- Increments the fetch address
- Maintains stacks
- Evaluates conditions
- Decrements the loop counter
- Calculates new addresses
- Maintains an instruction cache
- Handles interrupts

To accomplish these tasks, the sequencer uses the blocks shown in Figure 3-2. The sequencer's address multiplexer selects the value of the next fetch address from several possible sources. The fetched address enters the instruction pipeline, made up of the fetch address register, decode address register, and program counter (PC). These contain the 24-bit addresses of the instructions currently being fetched, decoded, and executed. The PC couples with the PC stack, which stores return addresses and top-of-loop addresses. All addresses generated by the sequencer are 24-bit program memory instruction addresses.

To manage events, the sequencer's interrupt controller handles interrupt processing, determines whether an interrupt is masked, and generates the appropriate interrupt vector address.

With selective caching, the instruction cache lets the DSP access data in program memory and fetch an instruction (from the cache) in the same cycle. The DAG2 data address generator outputs program memory data addresses.

The sequencer evaluates conditional instructions and loop termination conditions using information from the status registers. The loop address stack and loop counter stack support nested loops. The status stack stores status registers for implementing nested interrupt routines.

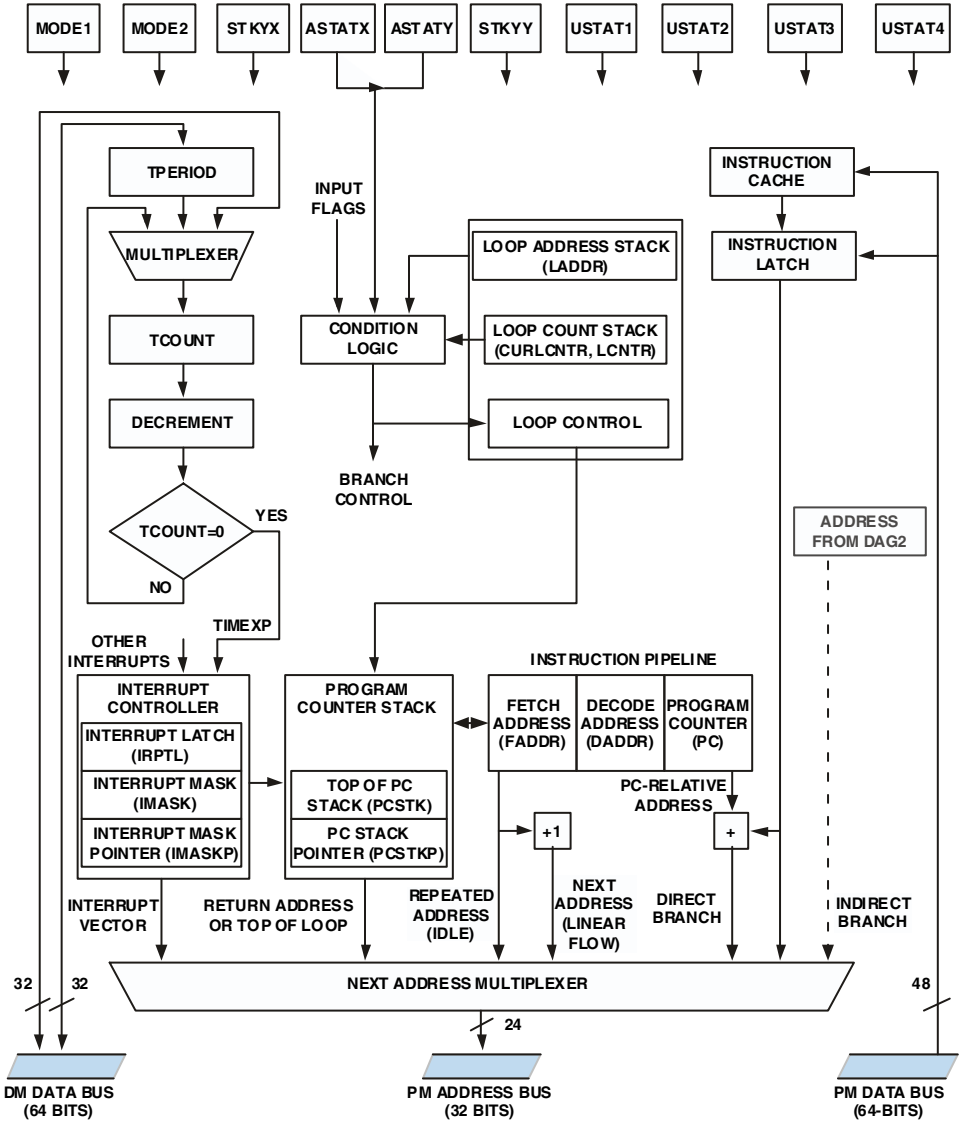


Figure 3-2. Program Sequencer Block Diagram

Table 3-1 and Table 3-2 list the registers within and related to the program sequencer. All registers in the program sequencer are universal registers, so they are accessible to other universal registers and to data memory. All the sequencer's registers and the tops of stacks are readable, and all these registers are writable, except for the fetch address, decode address, and PC. Pushing or popping the PC stack is done with a write to the PC stack pointer, which is readable and writable. Pushing or popping the loop address stack requires explicit instructions.

A set of system control registers configures or provides input to the sequencer. These registers appear across the top and within the interrupt controller of Figure 3-2. A bit manipulation instruction permits setting, clearing, toggling, or testing specific bits in the system registers. For information on this instruction (Bit), see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Writes to some of these registers do not take effect on the next cycle. For example, after a write to the `MODE1` register to enable ALU saturation mode, the change does not take effect until two cycles after the write. Also, some of these registers do not update on the cycle immediately following a write. It takes an extra cycle before a read of the register returns the new value.

With the lists of sequencer and system registers, Table 3-1 and Table 3-2 summarize the number of extra cycles (latency) for a write to take effect (effect latency) and for a new value to appear in the register (read latency). A "`0`" indicates that the write takes effect or appears in the register on the next cycle after the write instruction is executed, and a "`1`" indicates one extra cycle.

Table 3-1. Program Sequencer Registers Read and Effect Latencies

Register	Contents	Bits	Read Latency	Effect Latency
FADDR	fetch address	24	—	—
DADDR	decode address	24	—	—
PC	execute address	24	—	—
PCSTK	top of PC stack	24	0	0
PCSTKP	PC stack pointer	5	1	1
LADDER	top of loop address stack	32	0	0
CURLCNTR	top of loop count stack (current loop count)	32	0	0
LCNTR	loop count for next DO UNTIL loop	32	0	0

Table 3-2. System Registers Read and Effect Latencies[1]

Register	Contents	Bits	Read Latency	Maximum Effect Latency
MODE1	mode control bits	32	0	1
MODE2	mode control bits	32	0	1
IRPTL	interrupt latch	32	0	1
IMASK	interrupt mask	32	0	1
IMASKP	interrupt mask pointer (for nesting)	32	1	1
MMASK	mode mask	32	0	1
FLAGS	flag inputs	32	0	1
LIRPTL	link port interrupt latch/mask	32	0	1
ASTATX	arithmetic status flags	32	0	1
ASTATY	arithmetic status flags	32	0	1
STKYX	sticky status flags	32	0	1
STKYY	sticky status flags	32	0	1
USTAT1	user-defined status flags	32	0	0

Table 3-2. System Registers Read and Effect Latencies[1] (Cont'd)

Register	Contents	Bits	Read Latency	Maximum Effect Latency
USTAT2	user-defined status	32	0	0
USTAT3	user-defined status	32	0	0
USTAT4	user-defined status	32	0	0

1 The number of cycles it takes for the effect latencies for different registers (for example, MODE1) given above is just a maximum value. Different bits in these registers have different effect latencies ranging from 0 to the maximum value listed. Hence users can write code
– That does not have any dependency on the above effect latencies
– Such that there are delays up to the number of cycles specified in the above Maximum Effect Latency column

The following sections in this chapter explain how to use each of the functional blocks in Figure 3-2 on page 3-5.

- "Instruction Pipeline" on page 3-9
- "Instruction Cache" on page 3-10
- "Branches and Sequencing" on page 3-15
- "Loops and Sequencing" on page 3-21
- "Interrupts and Sequencing" on page 3-33
- "Timer and Sequencing" on page 3-50
- "Stacks and Sequencing" on page 3-52
- "Conditional Sequencing" on page 3-54
- "SIMD Mode and Sequencing" on page 3-57

Instruction Pipeline

The program sequencer determines the next instruction address by examining both the current instruction being executed and the current state of the processor. If no conditions require otherwise, the DSP executes instructions from program memory in sequential order by incrementing the fetch address. Using its instruction pipeline, the DSP processes instructions in three clock cycles:

- **Fetch cycle.** The DSP reads the instruction from either the on-chip instruction cache or from program memory.
- **Decode cycle.** The DSP decodes the instruction, generating conditions that control instruction execution.
- **Execute cycle.** The DSP executes the instruction; the operations specified by the instruction complete in a single cycle.

These cycles overlap in the pipeline, as shown in Table 3-3. In sequential program flow, when one instruction is being fetched, the instruction fetched in the previous cycle is being decoded, and the instruction fetched two cycles before is being executed. Sequential program flow always has a throughput of one instruction per cycle.

Table 3-3. Pipelined Execution Cycles

Cycles	Fetch	Decode	Execute
1	0x08 ↘		
2	0x09 ↘	0x08 ↘	
3	0x0A ↘	0x09 ↘	0x08
4	0x0B ↘	0x0A ↘	0x09
5	0x0C	0x0B	0x0A

Any non-sequential program flow can potentially decrease the DSP's instruction throughput. Non-sequential program operations include:

- Program memory data accesses that conflict with instruction fetches
- Jumps
- Subroutine calls and returns
- Interrupts and return
- Loops

Instruction Cache

Usually, the sequencer fetches an instruction from memory on each cycle. Occasionally, bus constraints prevent some of the data and instructions from being fetched in a single cycle. To alleviate these data flow constraints, the DSP has an instruction cache, which appears in Figure 3-2 on page 3-5. When the DSP executes an instruction that requires data access over the PM data bus, there is a bus conflict because the sequencer uses the PM data bus for fetching instructions. To avoid these conflicts, the DSP caches these instructions, reducing delays. Except for enabling or disabling the cache, its operation requires no user intervention. For more information, see "Using the Cache" on page 3-12.

The first time the DSP encounters a fetch conflict, the DSP must wait to fetch the instruction on the following cycle, causing a delay. The DSP automatically writes the fetched instruction to the cache to prevent the same delay from happening again. The sequencer checks the instruction cache on every program memory data access. If the instruction needed is in the cache, the instruction fetch from the cache happens in parallel with the program memory data access, without incurring a delay.

Because of the three-stage instruction pipeline, as the DSP executes instruction (at address n) that requires a program memory data access, this execution creates a conflict with the instruction fetch (at address n+2), assuming sequential execution. The cache stores the fetched instruction (n+2), not the instruction requiring the program memory data access.

If the instruction needed to avoid a conflict is in the cache, the cache provides the instruction while the program memory data access is performed. If the needed instruction is not in the cache, the instruction fetch from memory takes place in the cycle following the program memory data access, incurring one cycle of overhead. The fetched instruction is loaded into the cache, if the cache is enabled and not frozen, so that it is available the next time the same conflict occurs.

Figure 3-3 shows a block diagram of the instruction cache. The cache holds 32 instuction-address pairs. These pairs (or cache entries) are arranged into 16 (15–0) cache sets according to their address' 4 least significant bits (3–0). The two entries in each set (entry 0 and entry 1) have a valid bit, indicating whether the entry contains a valid instruction. The least recently used (LRU) bit for each set indicates which entry was not used last (0=entry 0 and 1=entry 1).

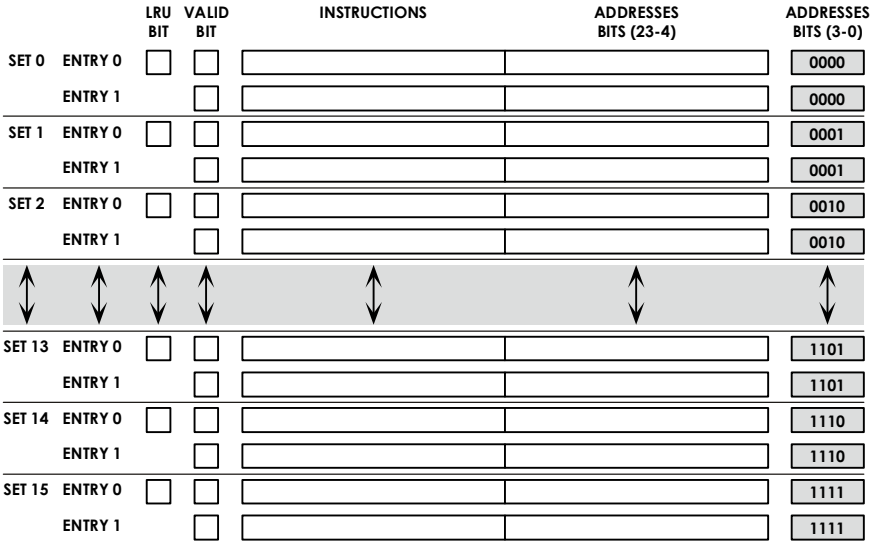


Figure 3-3. Instruction Cache Architecture

The cache places instructions in entries according to the 4 LSBs of the instruction's address. When the sequencer checks for an instruction to fetch from the cache, it uses the 4 address LSBs as an index to a cache set. Within that set, the sequencer checks the addresses of the two entries, looking for the needed instruction. If the cache contains the instruction, the sequencer uses the entry and updates the LRU bit (if necessary) to indicate the entry did not contain the needed instruction.

When the cache does not contain a needed instruction, the cache loads a new instruction and its address, placing these in the least recently used entry of the appropriate cache set and toggling the LRU bit (if necessary).

Using the Cache

After a DSP reset, the cache starts cleared (containing no instructions), unfrozen, and enabled. From then on, the MODE2 register controls the

operating mode of the instruction cache. Table A-3 on page A-7 lists all the bits in MODE2. The following bits in MODE2 control cache modes:

- **Cache Disable.** Bit 4 (CADIS) directs the sequencer to disable the cache (if 1) or enable the cache (if 0). Disabling the cache does not mark the current content of the cache as invalid. If the cache is to be enabled again, the existing content is used again. To clear the cache, use the FLUSH CACHE instruction.
- **Cache Freeze.** Bit 19 (CAFRZ) directs the sequencer to freeze the contents of the cache (if 1) or let new entries displace the entries in the cache (if 0).



If self-modifying code (e.g. software loader kernel) or software overlays are used, execute a FLUSH CACHE instruction followed by a NOP before executing the new code. Otherwise old content from the cache could still be used, although the code has changed.

When changing the cache's mode, note that an instruction containing a program memory data access must not be placed directly after a cache enable or cache disable instruction, because the DSP must wait at least one cycle before executing the PM data access. A program should have a NOP inserted after the cache enable instruction if necessary.

Optimizing Cache Usage

Usually, cache operation is efficient and requires no intervention, but certain ordering of instructions can work against the cache's architecture and can degrade cache efficiency. When the order of PM data accesses and instruction fetches continuously displaces cache entries and loads new entries, the cache is not being efficient. Rearranging the order of these instructions remedies this inefficiency.

An example of code that works against cache efficiency appears in Table 3-4.

Table 3-4. Cache-Inefficient Code

Address	Instruction
0x0100	lcntr=1024, do Outer until lce;
0x0101	r0=dm(i0,m0), pm(i8,m8)=f3;
0x0102	r1=r0-r15;
0x0103	if eq call (Inner);
0x0104	f2=float r1;
0x0105	f3=f2*f2;
0x0106	Outer: f3=f3+f4;
0x0107	pm(i8,m8)=f3;
...	
0x0200	Inner: r1=R13;
0x0201	r14=pm(i9,m9);
...	
0x0211	pm(i9,m9)=r12;
...	
0x021F	rts;

The program memory data access at address `0x101` in the loop, Outer, causes the cache to load the instruction at `0x103` (into set 3). Each time the program calls the subroutine, Inner, the program memory data accesses at `0x201` and `0x211` displace the instruction at `0x103` by loading the instructions at `0x203` and `0x213` (also into set 3). If the program only calls the Inner subroutine rarely during the Outer loop execution, the repeated cache loads do not greatly influence performance. If the program frequently calls the subroutine while in the loop, the cache inefficiency has a noticeable effect on performance.

To improve cache efficiency on this code (for instance, if the execution time of the Outer loop is critical), you should rearrange the order of some instructions. Moving the subroutine call up one location (starting at `0x201`) would work here, because with that order the two cached instructions end up in cache set 4 instead of set 3.

Branches and Sequencing

One of the types of non-sequential program flow that the sequencer supports is branching. A branch occurs when a Jump or Call/return instruction begins execution at a new location, other than the next sequential address. For descriptions on how to use the Jump and Call/return instructions, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Briefly, these instructions operate as follows.

- A Jump or a Call instruction transfers program flow to another memory location. The difference between a Jump and a Call is that a Call automatically pushes the return address (the next sequential address after the Call instruction) onto the PC stack. This push makes the address available for the Call instruction's matching return instruction to allow easy return from the subroutine.
- A return instruction causes the sequencer to fetch the instruction at the return address, which is stored at the top of the PC stack. The two types of return instructions are return from subroutine (RTS)

and return from interrupt (RTI). While the return from subroutine (RTS) only pops the return address off the PC stack, the return from interrupt (RTI) pops the return address and:

- Pops the status stack if the ASTATx,y and MODE1 status registers have been pushed for any of the following interrupts: IRQ2-0, timer, or VIRPT.
- Clears the interrupt's bit in the interrupt latch register (IRPTL) and the interrupt mask pointer (IMASKP).

You can specify a number of parameters for branches:

- Jump and Call/return instructions can be conditional. The program sequencer can evaluate status conditions to decide whether to execute a branch. If no condition is specified, the branch is always taken. For more information on these conditions, see "Conditional Sequencing" on page 3-54.
- Jump and Call/return instructions can be immediate or delayed. Because of the instructions pipeline, an immediate branch incurs two lost (overhead) cycles. A delayed branch has no overhead. For more information, see "Delayed Branches" on page 3-17.
- Jump instructions that appear within a loop or within an interrupt service routine have additional options. For information on the loop abort (LA) option, see "Loops and Sequencing" on page 3-21. For information on the loop re-entry (LR) option, see "Restrictions On Ending Loops" on page 3-23. For information on the clear interrupt (CI) option, see "Interrupts and Sequencing" on page 3-33.

The sequencer block diagram in Figure 3-2 on page 3-5 shows that branches can be direct or indirect. The difference is that the sequencer generates the address for a direct branch, and the PM data address generator (DAG2) produces the address for an indirect branch.

Direct branches are Jump or Call/return instructions that use an absolute—not changing at runtime—address (such as a program label) or use a PC-relative address. Some instruction examples that cause a direct branch are:

```
jump fft1024; {where fft1024 is an address label}
call (pc,10); {where (pc,10) a PC-relative address}
```

Indirect branches are Jump or Call/Return instructions that use a dynamic—changes at runtime—address that comes from the PM data address generator. For more information on the data address generator, see "Data Address Generators". Some instruction examples that cause an indirect branch are:

```
jump (m8,i12); {where (m8,i12) are DAG2 registers}
call (m9,i13); {where (m9,i13) are DAG2 registers}
```

Conditional Branches

The sequencer supports conditional branches. These are Jump or Call/return instructions whose execution is based on testing an If condition. For more information on condition types in If condition instructions, see "Conditional Sequencing" on page 3-54. Note that the DSP's Single-Instruction, Multiple-Data mode influences the execution of conditional branches. For more information, see "SIMD Mode and Sequencing" on page 3-57.

Delayed Branches

The instruction pipeline influences how the sequencer handles branches. For immediate branches—Jumps and Call/return instructions not specified as delayed branches (DB), two instruction cycles are lost (NOPs) as the pipeline empties and refills with instructions from the new branch.

As shown in Table 3-5 on page 3-18 and Table 3-6 on page 3-18, the DSP does not execute the two instructions after the branch, which are in

the fetch and decode stages. For a Call, the decode address (the address of the instruction after the Call) is the return address. During the two lost (no-operation) cycles, the pipeline fetches and decodes the first instruction at the branch address.

Table 3-5. Pipelined Execution Cycles For Immediate Branch (Jump/Call)

Cycles	Fetch	Decode	Execute
1	n+2	n+1→nop[1]	n
2	j[2]	n+2→nop[3]	NOP
3	j+1	j	NOP
4	j+2	j+1	j

Note that n is the branching instruction, and j is the instruction branch address
1. n+1 suppressed
2. For call, n+1 pushed on PC stack
3. n+2 suppressed

Table 3-6. Pipelined Execution Cycles For Immediate Branch (Return)

Cycles	Fetch	Decode	Execute
1	n+2	n+1→nop[1]	n[2]
2	r	n+2→nop[3]	NOP
3	r+1	r	NOP
4	r+2	r+1	r

Note that n is the branching instruction, and r is the instruction branch address
1. n+1 suppressed
2. r (n+1 in Table 3-5) popped from PC stack
3. n+2 suppressed

For delayed branches—Jumps and Call/return instructions with the delayed branches (DB) modifier, no instruction cycles are lost in the pipe-

line, because the DSP executes the two instructions after the branch while the pipeline fills with instructions from the new branch.

As shown in Table 3-7 and Table 3-8, the DSP executes the two instructions after the branch, while the instruction at the branch address is fetched and decoded. In the case of a Call, the return address is the third address after the branch instruction. While delayed branches use the instruction pipeline more efficiently than immediate branches, it is important to note that delayed branch code can be harder to understand because of the instructions between the branch instruction and the actual branch.

Table 3-7. Pipelined Execution Cycles For Delayed Branch (Jump or Call)

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n
2	j[1]	n+2	n+1
3	j+1	j	n+2
4	j+2	j+1	j

Note that n is the branching instruction, and j is the instruction branch address
1. For call, n+3 pushed on PC stack

Table 3-8. Pipelined Execution Cycles For Delayed Branch (return)

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n[1]
2	r	n+2	n+1
3	r+1	r	n+2
4	r+2	r+1	r

Note that n is the branching instruction, and r is the instruction branch address
1. r (n+3 in Table 3-7) popped from PC stack

Besides being somewhat more challenging to code, there are also some limitations on delayed branches that stem from the instruction pipeline

architecture. Because the delayed branch instruction and the two instructions that follow it must execute sequentially, the instructions in the two locations that follow a delayed branch instruction may not be any of the following:

- Other branches (no Jump, Call, or return instructions)
- Any manipulations of the PC stack, status stack or loop stacks
- Any writes to the PC stack pointer
- Any loops or other breaks in sequential operation (no Do/Until or Idle instructions)



Development software for the DSP should always flag these types of instructions in the two locations after a delayed branch instruction as code errors.



It is possible to follow a delayed branch instruction with a Jump, Call, or return instruction in one special case. If the sequential branch instructions use mutually exclusive conditions, one branch may follow another. The following example is valid.

```
if gt jump (PC, 7) (db); // if greater than...
if led jump (PC, 11) (db); // if less than or equal...
```

Interrupt processing is also influenced by delayed branches and the instruction pipeline. Because the delayed branch instruction and the two instructions that follow it must execute sequentially, the DSP does not immediately process an interrupt that occurs in between a delayed branch instruction and either of the two instructions that follow. Any interrupt that occurs during these instructions is latched, but not processed until the branch is complete.



During a delayed branch, a program can read the PC stack or PC stack pointer immediately after a delayed call or return, but this read shows that the return address on the PC stack has already been pushed or popped, even though the branch has not occurred yet.

Loops and Sequencing

Another type of non-sequential program flow that the sequencer supports is looping. A loop occurs when a Do/Until instruction causes the DSP to repeat a sequence of instructions until a condition tests true.

A special condition for terminating a loop is Loop Counter Expired (LCE). This condition tests whether the loop has completed the number of iterations in the LCNTR register. Loops that terminate with conditions other than LCE have some additional restrictions. For more information, see "Restrictions On Ending Loops" on page 3-23 and "Restrictions On Short Loops" on page 3-24. For more information on condition types in Do/Until instructions, see "Conditional Sequencing" on page 3-54.



The DSP's Single-Instruction, Multiple-Data mode influences the execution of loops. For more information, "SIMD Mode and Sequencing" on page 3-57.

The Do/Until instruction uses the sequencer's loop and condition features, which appear in Figure 3-2 on page 3-5. These features provide efficient software loops, without the overhead of additional instructions to branch, test a condition, or decrement a counter. The following code example shows a Do/Until loop that contains three instructions and iterates 30 times.

```
LCNTR=30, DO the_end UNTIL LCE; {loop iterates 30 times}
R0=DM(I0,M0), F2=PM(I8,M8);
R1=R0-R15;
the_end: F4=F2+F3;              {last instruction in loop}
```

When executing a Do/Until instruction, the program sequencer pushes the address of the loop's last instruction and loop's termination condition onto the loop address stack. The sequencer also pushes the top-of-loop address—address of the instruction following the Do/Until instruction—onto the PC stack.

The sequencer's instruction pipeline architecture influences loop termination. Because instructions are pipelined, the sequencer must test the termination condition (and, if the loop is counter-based, decrement the counter) before the end of the loop. Based on the test's outcome, the next fetch either exits the loop or returns to the top-of-loop.

The condition test occurs when the DSP is executing the instruction two locations before the last instruction in the loop (at location e – 2, where e is the end-of-loop address). If the condition tests false, the sequencer repeats the loop, fetching the instruction from the top-of-loop address, which is stored on the top of the PC stack. If the condition tests true, the sequencer terminates the loop, fetching the next instruction after the end of the loop and popping the loop and PC stacks.

A special case of loop termination is the loop abort instruction, Jump (LA). This instruction causes an automatic loop abort when it occurs inside a loop. When the loop aborts, the sequencer pops the PC and loop address stacks once. If the aborted loop was nested, the single pop of the stacks leaves the correct values in place for the outer loop.

Table 3-9 and Table 3-10 on page 3-23 show the pipeline states for loop iteration and termination.

Table 3-9. Pipelined Execution Cycles For Loop Back (Iteration)

Cycles	Fetch	Decode	Execute
1	e	e – 1	e – 2[1]
2	b[2]	e	e – 1
3	b+1	b	e
4	b+2	b+1	b

Note that e is the loop end instruction, and b is the loop start instruction
1. Termination condition tests false
2. Loop start address is top of PC stack

Table 3-10. Pipelined Execution Cycles For Loop Termination

Cycles	Fetch	Decode	Execute
1	e	e – 1	e – 2[1]
2	e+1[2]	e	e – 1
3	e+2	e+1	e
4	e+3	e+2	e+1

Note that e is the loop end instruction
1. Termination condition tests true
2. Loop aborts and loop stacks pop

Restrictions On Ending Loops

The sequencer's loop features (which optimize performance in many ways) limit the types of instructions that may appear at or near the end of the loop. These restrictions include:

Nested loops may not use the same end-of-loop instruction address.

- Nested loops with a non-counter-based loop as the outer loop must place the end address of the outer loop at least two addresses after the end address of the inner loop.
- Nested loops with a non-counter-based loop as the outer loop that use the loop abort instruction, Jump (LA), to abort the inner loop may not Jump (LA) to the last instruction of the outer loop.
- An instruction that writes to the loop counter from memory may not be used as the third-to-last instruction of a counter-based loop (at e – 2, where e is the end-of-loop address).

- An If Not LCE instruction may not be used as the instruction that follows a write to CURLCNTR from memory.
- Branch (Jump or Call/return) instructions may not be used as any of the last three instructions of a loop. This no end-of-loop branches rule also applies to single-instruction and two-instruction loops with only one iteration.

There is one exception to the no end-of-loop branches rule. The last three instructions of a loop may contain an immediate Call—a Call without a DB modifier—that is paired with a loop re-entry return—a return (RTS) with loop re-entry modifier (LR). The immediate Call may be one of the last three instructions of a loop, but not in a one-instruction loop or a two-instruction, single-iteration loop.

Restrictions On Short Loops

The sequencer's pipeline features (which optimize performance in many ways) restrict how short loops iterate and terminate. Short loops (1- or 2-instruction loops) terminate in a special way because they are shorter than the instruction pipeline. Counter-based loops (Do/Until LCE) of one or two instructions are not long enough for the sequencer to check the termination condition two instructions from the end of the loop. In these short loops, the sequencer has already looped back when the termination condition is tested. The sequencer provides special handling to avoid overhead (NOP) cycles if the loop is iterated a minimum number of times.

Table 3-11 on page 3-25 and Table 3-12 on page 3-26 show the pipeline execution for counter-based single-instruction loops.

Table 3-11. Pipelined Execution Cycles for Single Instruction Counter-Based Loop With Three Iterations

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n[1]
2	n+1[2]	n+1	n+1 (pass 1)
3	n+2[3]	n+1	n+1 (pass 2)
4	n+3	n+2	n+1 (pass 3)
5	n+4	n+3	n+2

Note: n is the loop start instruction, and n+2 is the instruction after the loop
1. Loop count (LCNTR) equals 3
2. No opcode latch or fetch address update; count expired tests true
3. Loop iteration aborts; PC and loop stacks pop

Table 3-12. Pipelined Execution Cycles for Single Instruction Counter-Based Loop With Two Iterations (Two Overhead Cycles)

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n[1]
2	n+1[2]	n+1	n+1 (pass 1)
3	n+1[3]	n+1→nop[4]	n+1 (pass 2)
4	n+2	n+1→nop[5]	NOP
5	n+3	n+2	NOP
6	n+4	n+3	n+2

Note: n is the loop start instruction, and n+2 is the instruction after the loop
1. Loop count (LCNTR) equals 2
2. No opcode latch or fetch address update
3. Count expired tests true
4. Loop iteration aborts; PC and loop stacks pop; n+1 suppressed
5. n+1 suppressed

Table 3-13 and Table 3-14 on page 3-28 show the pipeline execution for counter-based two-instruction loops. For no overhead, a loop of length one must be executed at least three times and a loop of length two must be executed at least twice. Loops of length one that iterate only once or twice and loops of length two that iterate only once incur two cycles of overhead, because there are two aborted instructions after the last iteration to clear the instruction pipeline.

Table 3-13. Pipelined Execution Cycles For Two Instruction Counter-Based Loop With Two Iterations

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n[1]
2	n+1[2]	n+2	n+1 (pass 1)
3	n+2[3]	n+1	n+2 (pass 1)
4	n+3[4]	n+2	n+1 (pass 2)
5	n+4	n+3	n+2 (pass 2)
6	n+5	n+4	n+3

Note: n is the loop start instruction, and n+3 is the instruction after the loop
1. Loop count (LCNTR) equals 2
2. PC stack supplies loop start address
3. Count expired tests true
4. Loop iteration aborts; PC and loop stacks pop

Table 3-14. Pipelined Execution Cycles For Two Instruction Counter-Based Loop With One Iteration (Two Overhead Cycles)

Cycles	Fetch	Decode	Execute
1	n+2	n+1	n[1]
2	n+1[2]	n+2	n+1 (pass 1)
3	n+2[3]	n+1→nop[4]	n+2 (pass 1)
4	n+3	n+2→nop[5]	NOP
5	n+4	n+3	NOP
6	n+5	n+4	n+3

Note: n is the loop start instruction, and n+3 is the instruction after the loop
1. Loop count (LCNTR) equals 1
2. PC stack supplies loop start address
3. Count expired tests true
4. Loop iteration aborts; PC and loop stacks pop; n+1 suppressed
5. n+2 suppressed

Processing of an interrupt that occurs during the last iteration of a one-instruction loop that executes once or twice, a two-instruction loop that executes once, or the cycle following one of these loops (which is a NOP) is delayed by one cycle. Similarly, in a one-instruction loop that iterates at least three times, processing is delayed by one cycle if the interrupt occurs during the third-to-last iteration. For more information on pipeline execution during interrupts, see “Interrupts and Sequencing” on page 3-33.

Short non-counter-based loops terminate differently from short counter-based loops. These differences stem from the architecture of the pipeline and conditional logic.

- In a three-instruction non-counter-based loop, the sequencer tests the termination condition when the DSP executes the top of loop instruction. When the condition tests true, the sequencer completes the iteration of the loop and terminates.
- In a two-instruction non-counter-based loop, the sequencer tests the termination condition when the DSP executes the last (second) instruction. If the condition becomes true when the first instruction is executed, the condition tests true during the second instruction, and the sequencer completes one more iteration of the loop before exiting. If the condition becomes true during the second instruction, the sequencer completes two more iterations of the loop before exiting.
- In a one-instruction non-counter-based loop, the sequencer tests the termination condition every cycle. After the cycle when the condition becomes true, the sequencer completes three more iterations of the loop before exiting.

Loop Address Stack

The sequencer's loop support, which appears in Figure 3-2 on page 3-5, includes a loop address stack. The loop address stack is six levels deep by 32 bits wide.

The LADDR register contains the top entry on the loop address stack. This register is readable and writable over the DM Data bus. Reading and writing LADDR does not move the loop address stack pointer; only a stack push or pop, performed with explicit instructions, moves the stack pointer. LADDR contains the value 0xFFFF FFFF when the loop address stack is empty. Table A-13 on page A-30 lists all the bits in LADDR.

The sequencer pushes an entry onto the loop address stack when executing a Do/Until or Push Loop instruction. The stack entry pops off the stack two instructions before the end of its loop's last iteration or on a Pop Loop instruction. A stack overflow occurs if a seventh entry (one more than full) is pushed onto the loop stack. The stack is empty when no entries are occupied.

The loop stacks' overflow or empty status is available. Because the sequencer keeps the loop stack and loop counter stack synchronized, the same overflow and empty flags apply to both stacks. These flags are in the sticky status register (STKYx). For more information on STKYx, see Table A-5 on page A-13. For more information on how these flags work with the loop stacks, see "Loop Counter Stack". Note that a loop stack overflow causes a maskable interrupt.

Because the sequencer tests the termination condition two instructions before the end of the loop, the loop stack pops before the end of the loop's final iteration. If a program reads LADDR at either of these instructions, the value is already the termination address for the next loop stack entry.

Loop Counter Stack

The sequencer's loop support, which appears in Figure 3-2 on page 3-5, includes a loop counter stack. The sequencer keeps the loop counter stack synchronized with the loop address stack, with both stacks always having the same number of locations occupied. Because these stacks are synchronized, the same empty and overflow status flags from the STKYx register apply to both stacks.

The loop counter stack is six locations deep. The stack is full when all entries are occupied, is empty when no entries are occupied, and is overflowed if a push occurs when the stack is already full. Bits in the STKYx register indicate the loop counter stack full and empty states. Table A-5

on page A-13 lists the bits in the STYKx register. The STKYx bits that indicate loop counter stack status are:

- **Loop stacks overflowed.** Bit 25 (LSOV) indicates that the loop counter stack and loop stack are overflowed (if 1) or not overflowed (if 0)—A sticky bit
- **Loop stacks empty.** Bit 26, (LSEM) indicates that the loop counter stack and loop stack are empty (if 1) or not empty (if 0)—Not sticky, cleared by a Push

Within the sequencer, the current loop counter (CURLCNTR) and loop counter (LCNTR) registers allow access to the loop counter stack. CURLCNTR tracks iterations for a loop being executed, and LCNTR holds the count value before the loop is executed. The two counters let the DSP maintain the count for an outer loop, while a program is setting up the count for an inner loop.

The top entry in the loop counter stack (CURLCNTR) always contains the current loop count. This register is readable and writable over the DM Data bus. Reading CURLCNTR when the loop counter stack is empty returns the value 0xFFFF FFFF.

The sequencer decrements the value of CURLCNTR for each loop iteration. Because the sequencer tests the termination condition two instruction cycles before the end of the loop, the loop counter also decrements before the end of the loop. If a program reads CURLCNTR at either of the last two loop instructions, the value is already the count for the next iteration.

The loop counter stack pops two instructions before the end of the last loop iteration. When the loop counter stack pops, the new top entry of the stack becomes the CURLCNTR value—the count in effect for the executing loop. If there is no executing loop, the value of CURLCNTR is 0xFFFF FFFF after the pop.

Writing CURLCNTR does not cause a stack push. If a program writes a new value to CURLCNTR, it changes the count value of the loop currently execut-

ing. When no Do/Until LCE loop is executing, writing to CURLCNTR has no effect. Because the processor must use CURLCNTR to perform counter-based loops, there are some restrictions on how a program can write CURLCNTR. For more information, see "Restrictions On Ending Loops" on page 3-23.

The next-to-top entry in the loop counter stack (LCNTR) is the location on the stack that takes effect on the next loop stack push. To set up a count value for a nested loop without changing the count for the currently executing loop, a program writes the count value to LCNTR.



A value of zero in LCNTR causes a loop to execute 2^{32} times.

A Do/Until LCE instruction pushes the value of LCNTR onto the loop count stack, making that value the new CURLCNTR value. Figure 3-4 on page 3-33 demonstrates this process for a set of nested loops. The previous CURLCNTR value is preserved one location down in the stack. If a program reads LCNTR when the loop counter stack is full, the stack returns invalid data. When the loop counter stack is full, the stack discards any data written to LCNTR. If a program reads LCNTR during the last two instructions of a terminating loop, the value of LCNTR is the last CURLCNTR value for the loop.

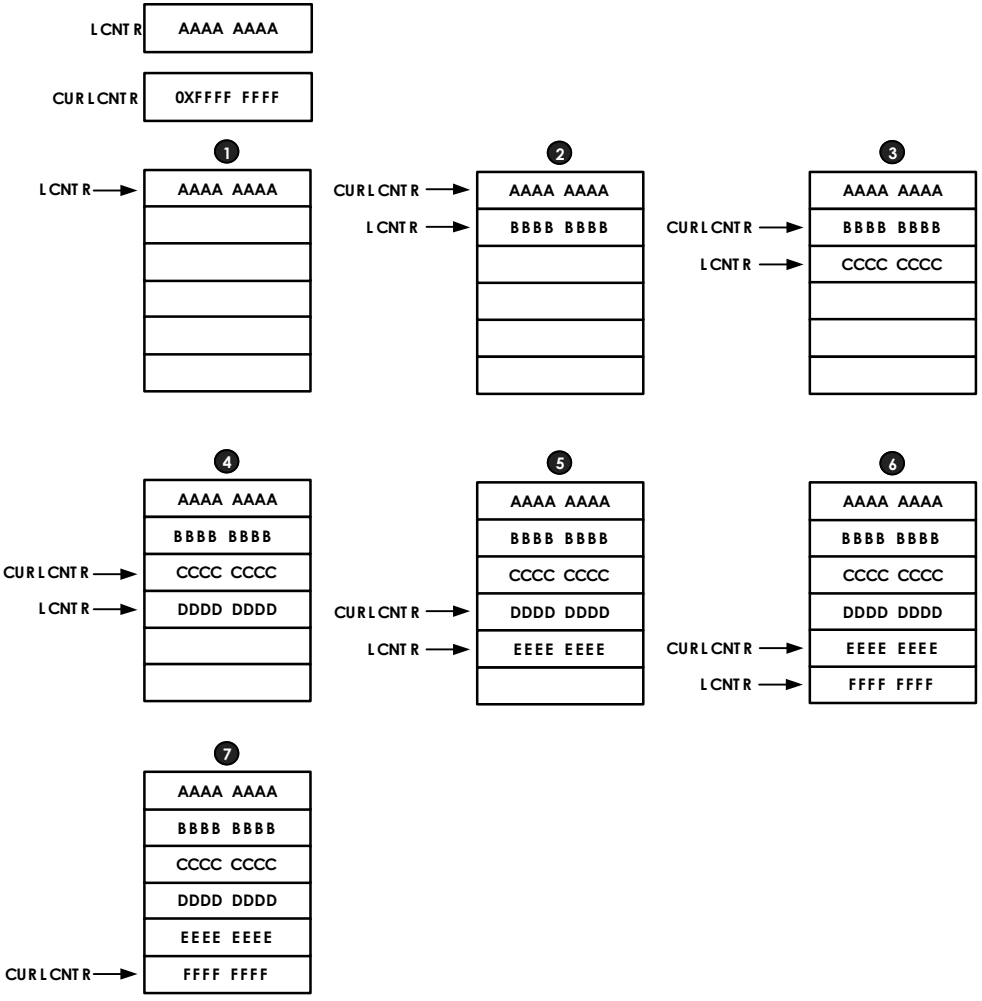


Figure 3-4. Pushing the Loop Counter Stack for Nested Loops

Interrupts and Sequencing

Another type of non-sequential program flow that the sequencer supports is interrupt processing. Interrupts may stem from a variety of conditions,

both internal and external to the processor. In response to an interrupt, the sequencer processes a subroutine call to a predefined address, the interrupt vector. The DSP assigns a unique vector to each type of interrupt.

The DSP supports three prioritized, individually-maskable external interrupts, each of which can be either level- or edge-sensitive. External interrupts occur when another device asserts one of the DSP's interrupt inputs ($\overline{\texttt{IRQ2-0}}$). The DSP also supports internal interrupts. An internal interrupt can stem from arithmetic exceptions, stack overflows, or circular data buffer overflows. Several factors control the DSP's response to an interrupt. The DSP responds to an interrupt request if:

- The DSP is executing instructions or is in an Idle state
- The interrupt is not masked
- Interrupts are globally enabled
- A higher priority request is not pending

When the DSP responds to an interrupt, the sequencer branches program execution with a Call to the corresponding interrupt vector address. Within the DSP's program memory, the interrupt vectors are grouped in an area called the interrupt vector table. The interrupt vectors in this table are spaced at 4-instruction intervals. For a list of interrupt vector addresses and their associated latch and mask bits, see Table B-1 on page B-1. Each interrupt vector has associated latch and mask bits. Table A-9 on page A-18 lists the latch and mask bits.

To process an interrupt, the DSP's program sequencer does the following.

1. Outputs the appropriate interrupt vector address
2. Pushes the current PC value (the return address) on to the PC stack
3. Pushes the current value of the `ASTATx,y` and `MODE1` registers onto the status stack (if the interrupt is $\overline{\texttt{IRQ2-0}}$, timer, or `VIRPT`)

4. Sets the appropriate bit in the interrupt latch register (IRPTL)
5. Alters the interrupt mask pointer (IMASKP) to reflect the current interrupt nesting state, depending on the nesting mode

At the end of the interrupt service routine, the sequencer processes the return from interrupt (RTI) instruction and does following.

1. Returns to the address stored at the top of the PC stack
2. Pops this value off of the PC stack
3. Pops the status stack (if the ASTATx,y and MODE1 status registers were pushed for the $\overline{\text{IRQ2-0}}$, timer, or VIRPT interrupt)
4. Clears the appropriate bit in the interrupt latch register (IRPTL) and interrupt mask pointer (IMASKP)

Except for reset, all interrupt service routines should end with a return-from-interrupt (RTI) instruction. After reset, the PC stack is empty, so there is no return address. The last instruction of the reset service routine should be a jump to the start of your program.

If software writes to a bit in IRPTL forcing an interrupt, the processor recognizes the interrupt in the following cycle, and two cycles of branching to the interrupt vector follow the recognition cycle.

The DSP responds to interrupts in three stages: synchronization and latching (1 cycle), recognition (1 cycle), and branching to the interrupt

vector (2 cycles). Table 3-15, Table 3-16 on page 3-37, and Table 3-17 on page 3-38 show the pipelined execution cycles for interrupt processing.

Table 3-15. Pipelined Execution cycles For Interrupt During Single-Cycle Instruction

Cycles	Fetch	Decode	Execute
1	n + 1	n	n – 1[1]
2	n+ 2[2]	n+1→nop[3]	n
3	v[4]	n+2→NOP[5]	NOP
4	v+1	v	NOP
5	v+2	v+1	v

Note that n is the single-cycle instruction, and v is the interrupt vector instruction
1. Interrupt occurs
2. Interrupt recognized
3. n+1 pushed on PC stack; n+1 suppressed
4. Interrupt vector output
5. n+2 suppressed

Table 3-16. Pipelined Execution Cycles For Interrupt During Instruction With Conflicting PM Data Access (Instruction Not Cached)

Cycles	Fetch	Decode	Execute
1	n+1	n	n – 1[1]
2	—[2]	n+1→nop[3]	n
3	n+2[4]	n+1→nop[5]	NOP
4	v[6]	n+2→NOP[7]	NOP
5	v+1	v	NOP
6	v+2	v+1	v

Note that n is the conflicting instruction, and v is the interrupt vector instruction
1. Interrupt occurs
2. Interrupt recognized, but not processed; PM data access
3. n+1 suppressed
4. Interrupt processed
5. n+1 suppressed
6. Interrupt vector output
7. n+1 pushed on PC stack; n+2 suppressed

Table 3-17. Pipelined Execution Cycles for Interrupt During Delayed Branch Instruction

Cycles	Fetch	Decode	Execute
1	n+1	n	n – 1[1]
2	n+2[2]	n+1	n
3	j	n+2	n+1
4	j+1[3]	j→nop[4]	n+2
5	v[5]	j+1→NOP[6]	NOP
6	v+1	v	NOP
7	v+2	v+1	v

Note that n is the delayed branch instruction, j is the instruction at the branch address, and v is the interrupt vector instruction
1. Interrupt occurs
2. Interrupt recognized, but not processed
3. Interrupt processed
4. For a Call, n+3 (return address) is pushed onto the PC stack; j suppressed
5. Interrupt vector output
6. j pushed on PC stack; j+1 suppressed

For most interrupts, internal and external, only one instruction is executed after the interrupt occurs (and before the two instructions aborted) while the processor fetches and decodes the first instruction of the service routine. Because of the one-cycle delay between an arithmetic exception and the `STKYx,y` register update, there are two cycles after an arithmetic exception occurs before interrupt processing starts. Table 3-18 on page 3-39 lists the latency associated with the $\overline{\texttt{IRQ2-0}}$ interrupts and the multiprocessor vector interrupt.

Table 3-18. Minimum Latency of the $\overline{\text{IRQ2-0}}$ and VIRPT Interrupts

Interrupt	Minimum Latency
$\overline{\text{IRQ2-0}}$	3 cycles
VIRPT	6 cycles

If nesting is enabled and a higher priority interrupt occurs immediately after a lower priority interrupt, the service routine of the higher priority interrupt is delayed by one additional cycle. This delay allows the first instruction of the lower priority interrupt routine to be executed before it is interrupted. For more information, "Nesting Interrupts" on page 3-45.

Certain DSP operations that span more than one cycle hold off interrupt processing. If an interrupt occurs during one of these operations, the DSP latches the interrupt, but delays its processing. The operations that have delayed interrupt processing are as follows.

- A branch (Jump or Call/return) instruction and the following cycle, whether it is an instruction (in a delayed branch) or a NOP (in a non-delayed branch)
- The first of the two cycles used to perform a program memory data access and an instruction fetch when the instruction is not cached
- The third-to-last iteration of a one-instruction loop
- The last iteration of a one-instruction loop executed once or twice or of a two-instruction loop executed once, and the following cycle (which is a NOP)
- The first of the two cycles used to fetch and decode the first instruction of an interrupt service routine

- Any waitstates for external memory accesses
- Any external memory access that is required when the DSP does not have control of the external bus, during a host bus grant or when the DSP is a bus slave in a multiprocessing system

Sensing Interrupts

The DSP supports two types of interrupt sensitivity—the signal shape that triggers the interrupt. On interrupt pins (IRQ2-0), either the input signal's edge or level can trigger an external interrupt.

The DSP detects a level-sensitive interrupt if the signal input is low (active) when sampled on the rising edge of CLKIN. A level-sensitive interrupt must go high (inactive), before the processor returns from the interrupt service routine. If a level-sensitive interrupt is still active when the DSP samples it after returning from its service routine, the DSP treats the signal as a new request, repeating the same interrupt routine without returning to the main program, assuming no higher priority interrupts are active.

The DSP detects an edge-sensitive interrupt if the input signal is high (inactive) on one cycle and low (active) on the next cycle when sampled on the rising edge of CLKIN. An edge-sensitive interrupt signal can stay active indefinitely without triggering additional interrupts. To request another interrupt, the signal must go high, then low again.

Edge-sensitive interrupts require less external hardware compared to level-sensitive requests, because there is never a need to negate the request. An advantage of level-sensitive interrupts is that multiple interrupting devices may share a single level-sensitive request line on a wired-OR basis, allowing for easy system expansion.

The MODE2 register controls external interrupt sensitivity. Table A-3 on page A-7 lists all bits in the MODE2 register.

The following bits in MODE2 control interrupt sensitivity.

- **Interrupt 0 Sensitivity.** Bit 0, (IRQ0E), directs the DSP to detect $\overline{\text{IRQ0}}$ as edge-sensitive (if 1) or level-sensitive (if 0).
- **Interrupt 1 Sensitivity.** Bit 1, (IRQ1E), directs the DSP to detect $\overline{\text{IRQ1}}$ as edge-sensitive (if 1) or level-sensitive (if 0).
- **Interrupt 2 Sensitivity.** Bit 2, (IRQ2E), directs the DSP to detect $\overline{\text{IRQ2}}$ as edge-sensitive (if 1) or level-sensitive (if 0).

The DSP accepts external interrupts that are asynchronous to the DSP's clock (CLKIN), allowing external interrupt signals to change at any time. An external interrupt must be held low at least one CLKIN cycle to guarantee that the DSP samples the signal.



External interrupts must meet the setup and hold time requirements relative to the rising edge of CLKIN. For information on interrupt signal timing requirements, see the DSP's data sheet.

Masking Interrupts

The sequencer supports interrupt masking—latching an interrupt, but not responding to it. Except for the $\overline{\text{RESET}}$ and $\overline{\text{EMU}}$ interrupts, all interrupts are maskable. If a masked interrupt is latched, the DSP responds to the latched interrupt if it is later unmasked.

Interrupts can be masked globally or selectively. Bits in the MODE1, IMASK, and LIRPTL registers control interrupt masking. Table A-2 on page A-3 lists the bits in MODE1, Table A-9 on page A-18 lists the bits in IMASK, and Table A-10 on page A-24 lists the bits in LIRPTL.

These bits control interrupt masking as follows.

- **Global interrupt enable.** MODE1, Bit 12, (IRPTEN), directs the DSP to enable (if 1) or disable (if 0) all interrupts.
- **Selective interrupt enable.** IMASK, Bits 32-0, direct the DSP to enable (if 1) or disable/mask (if 0) the corresponding interrupt.
- **Selective link port interrupt enable.** LIRPTL, Bits 21-16, (LPxMSK) direct the DSP to enable (if 1) or disable/mask (if 0) the corresponding link port interrupt.

Except for the non-maskable interrupts and boot interrupts, all interrupts are masked at reset. For booting, the DSP automatically unmasks and uses either the external port (EPOI) or link port (LP4I) interrupt after reset, depending on whether the ADSP-21160 DSP is booting from EPROM, host, or link ports.

Latching Interrupts

When the DSP recognizes an interrupt, the DSP's interrupt latch (IRPTL and LIRPTL) registers latch the interrupts—set a bit to record that the interrupt occurred. The bits in these registers indicate all interrupts that are currently being serviced or are pending. Because these registers are readable and writable, any interrupt (except reset) can be set or cleared in software. Note that writing to the reset bit (bit 1) in IRPTL puts the processor into an illegal state.

When an interrupt occurs, the sequencer sets the corresponding bit in IRPTL or LIRPTL. During execution of the interrupt's service routine, the DSP keeps this bit cleared—clearing the bit during every cycle to prevent the same interrupt from being latched while its service routine is executing. After the return from interrupt (RTI), the sequencer stops clearing the latch bit.

If necessary, it is possible to re-use an interrupt while it is being serviced. For more information, see "Reusing Interrupts" on page 3-47.

The interrupt latch bits in IRPTL correspond to interrupt mask bits in the IMASK register. In both registers, the interrupt bits are arranged in order of priority. The interrupt priority is from 0 (highest) to 31 (lowest). Interrupt priority determines which interrupt is serviced first when more than one occurs in the same cycle. Priority also determines which interrupts are nested when the DSP has interrupt nesting enabled. For more information, see "Nesting Interrupts" on page 3-45.

While IRPTL latches interrupts for a variety of events, the LIRPTL register contains latch and mask bits only for Link Port DMA interrupts. A logical Or'ing of link port interrupts (masked-latch state) appears in the LPSUM bit in the IRPTL register. Because the LPSUM bit has a corresponding mask bit in the IMASK register, programs can use LPSUM for a second level of link port interrupt masking.

Multiple events can cause arithmetic interrupts—fixed-point overflow (FIXI) and floating-point overflow (FLTOI), underflow (FLTUI), and invalid operation (FLTII). To determine which event caused the interrupt, a program can read the arithmetic status flags in the STYKx or STKYy status registers. Table A-5 on page A-13 lists the bits in these registers. Service routines for arithmetic interrupts must clear the appropriate STKYx or STKYy bits to clear the interrupt. If the bits are not cleared, the interrupt is still active after the return from interrupt (RTI).



Status bits in STKYy only apply in SIMD mode. For more information, see "Secondary Processing Element (PEy)" on page 2-36.

One event can cause multiple interrupts. The timer decrementing to zero causes two timer expired interrupts, TMZHI (high priority) and TMZLI (low priority). This feature allows selection of the priority for the timer interrupt. Programs should unmask the timer interrupt with the desired priority and leave the other one masked. If both interrupts are unmasked, IRPTL latches both interrupts when the timer reaches zero, and the DSP

services the higher priority interrupt first, then the lower priority interrupt.

The IRPTL also supports software interrupts. When a program sets the latch bit for one of these interrupts (SFT0I, SFT1I, SFT2I, or SFT3I), the sequencer services the interrupt, and the DSP branches to the corresponding interrupt routine. Software interrupts have the same behavior as all other maskable interrupts.

Stacking Status During Interrupts

To run in an interrupt driven system, programs depend on the DSP being restored to its pre-interrupt state after an interrupt is serviced. The sequencer's status stack eases the return from interrupt process by eliminating some interrupt service overhead—register saves and restores.

The status stack is fifteen locations deep. The stack is full when all entries are occupied, is empty when no entries are occupied, and is overflowed if a push occurs when the stack is already full. Bits in the STKYx register indicate the status stack full and empty states. Table A-5 on page A-13 lists the bits in the STYKx register. The STKYx bits that indicate status stack status are:

- **Status stack overflow.** Bit 23, (SSOV), indicates that the status stack is overflowed (if 1) or not overflowed (if 0)—a sticky bit.
- **Status stack empty.** Bit 24, (SSEM), indicates that the status stack is empty (if 1) or not empty (if 0)—not sticky, cleared by a Push.

For some interrupts ($\overline{\text{IRQ2-0}}$, timer expired, and VIRPT), the sequencer automatically pushes the ASTATx, ASTATy, and MODE1 registers onto the status stack. When the sequencer pushes an entry onto the status stack, the DSP uses the MMASK register to set up MODE1 register.

The sequencer automatically pops the ASTATx, ASTATY, and MODE1 registers from the status stack during the return from interrupt instruction (RTI).

In one other case—Jump (CI), the sequencer pops the stack. For more information, see "Reusing Interrupts" on page 3-47.

Only the $\overline{\text{IRQ2-0}}$, timer expired, and VIRPT interrupts cause the sequencer to push an entry onto the status stack. All other interrupts require explicit saves and restores of effected registers or require an explicit push or pop of the stack (Push/Pop Sts).

Pushing ASTATx, ASTATy, and MODE1 preserves the status and control bit settings, allowing a service routine to alter these bits with the knowledge that the original settings are automatically restored upon the return from the interrupt.

The top of the status stack contains the current values of ASTATx, ASTATy, and MODE1. Reading and writing these registers does not move the stack pointer. Explicit Push or Pop instructions do move the status stack pointer.

Nesting Interrupts

The sequencer supports interrupt nesting—responding to another interrupt while a previous interrupt is being serviced. Bits in the MODE1, IMASKP, and LIRPTL registers control interrupt nesting. Table A-2 on page A-3 lists the bits in MODE1, Table A-9 on page A-18 lists the bits in IMASKP, and Table A-10 on page A-24 lists the bits in LIRPTL.

These bits control interrupt nesting as follows.

- **Interrupt nesting enable.** MODE1, Bit 11 (NESTM), directs the DSP to enable (if 1) or disable (if 0) interrupt nesting.
- **Interrupt Mask Pointer.** IMASKP, 32 Bits, lists the interrupts in priority order and provides a temporary interrupt mask for each nesting level.
- **Link Port DMA Interrupt Mask Pointer.** LIRPTL, Bits 21-16, (LPxMSK), lists link port DMA interrupts in priority order and provides a temporary interrupt mask for each nesting level.

When interrupt nesting is disabled, a higher priority interrupt can not interrupt a lower priority interrupt's service routine. Other interrupts are latched as they occur, but the DSP processes them after the active routine finishes.

When interrupt nesting is enabled, a higher priority interrupt can interrupt a lower priority interrupt's service routine. Lower interrupts are latched as they occur, but the DSP process them after the nested routines finish.

Programs should only change the interrupt nesting enable (NESTM) bit while outside of an interrupt service routine or during the reset service routine.

If nesting is enabled and a higher priority interrupt occurs immediately after a lower priority interrupt, the service routine of the higher priority interrupt is delayed by one cycle. This delay allows the first instruction of the lower priority interrupt routine to be executed, before it is interrupted.

When servicing nested interrupts, the DSP uses the interrupt mask pointer (IMASKP) to create a temporary interrupt mask for each level of interrupt nesting; the IMASK value is not effected. The DSP changes

IMASKP each time a higher priority interrupt interrupts a lower priority service routine.

The bits in IMASKP correspond to the interrupts in order of priority. When an interrupt occurs, the DSP sets its bit in IMASKP. If nesting is enabled, the DSP uses IMASKP to generate a new temporary interrupt mask, masking all interrupts of equal or lower priority to the highest priority bit set in IMASKP and keeping higher priority interrupts the same as in IMASK. When a return from an interrupt service routine (RTI) is executed, the DSP clears the highest priority bit set in IMASKP and generates a new temporary interrupt mask, masking all interrupts of equal or lower priority to the highest priority bit set in IMASKP. The bit set in IMASKP that has the highest priority always corresponds to the priority of the interrupt being serviced.

If an interrupt re-occurs while its service routine is running and nesting is enabled, the DSP updates IRPTL, but does not service the interrupt. The DSP waits until the return from interrupt (RTI) completes, before vectoring to the service routine again.

If nesting is not enabled, the DSP masks out all interrupts and IMASKP is not used, but the DSP still updates IMASKP to create a temporary interrupt mask.



The interrupt controller uses the IMASKP register and the LPxMSKP bits of the LIRPTL register. These bits should not be modified to ensure proper functioning of the interrupt controller.

Reusing Interrupts

Unless interrupt nesting is enabled, the DSP ignores and does not latch an interrupt that re-occurs while its service routine is executing. When the interrupt initially occurs, the sequencer sets the corresponding bit in IRPTL. During execution of the service routine, the sequencer keeps this bit cleared—the DSP clears the bit during every cycle, preventing the same interrupt from being latched while its service routine is already executing.

If necessary, it is possible to re-use an interrupt while it is being serviced. Using a Jump clear interrupt—Jump (CI)—instruction in the interrupt service routine clears the interrupt, allowing its re-usage while the service routing is executing.

The Jump (CI) instruction reduces an interrupt service routine to a normal subroutine, clearing the appropriate bit in the interrupt latch and interrupt mask pointer and popping the status stack. After the Jump (CI) instruction, the DSP stops automatically clearing the interrupt's latch bit, allowing the interrupt to latch again.

When returning from a subroutine entered with a Jump (CI) instruction, a program must use a return loop re-entry—RTS (LR)—instruction. For more information, see "Restrictions On Ending Loops" on page 3-23.

The following example shows an interrupt service routine that is reduced to a subroutine with the (CI) modifier:

```
instr1; {interrupt entry from main program}
JUMP(PC,3) (DB,CI); {clear interrupt status}
instr3;
instr4;
instr5;
RTS (LR); {use LR modifier with return from subroutine}
```



The Jump(PC,3)(DB,CI) instruction actually only continues linear execution flow by jumping to the location PC + 3 (`instr5`), with the two intervening instructions (`instr3`, `instr4`) being executed because of the delayed branch (DB). This Jump instruction is only an example—a Jump (CI) can be to any location.

Interrupting IDLE

The sequencer supports placing the DSP in Idle—a special instruction that halts the processor core in a low-power state, until an external interrupt ($\overline{\texttt{IRQ2-0}}$), timer interrupt, DMA interrupt, or `VIRPT` vector interrupt

occurs. When executing an Idle instruction, the sequencer fetches one more instruction at the current fetch address and then suspends operation. The DSP's I/O processor is not effected by the Idle instruction—DMA transfers to or from internal memory continues uninterrupted.

The processor's internal clock and timer (if it is enabled) continue to run during Idle. When an external interrupt ($\overline{\text{IRQ2-0}}$), timer interrupt, DMA interrupt, or VIRPT vector interrupt occurs, the processor responds normally. After two cycles used to fetch and decode the first instruction of the interrupt service routine, the processor continues executing instructions normally.

Multiprocessing Interrupts

The sequencer supports a multiprocessor vector interrupt. The vector interrupt (VIRPT) permits passing interprocessor commands in multiple-processor systems. This interrupt occurs when an external processor (a host or another DSP) writes an address to the VIRPT register, inserting a new vector address for VIRPT.

There is room in the VIRPT register for the vector address and data for the service routine. Table A-18 on page A-48 lists the bits in the VIRPT registers.

When servicing a VIRPT interrupt, the DSP automatically pushes the status stack and executes the service routine located at the address specified in VIRPT. During the return from interrupt (RTI), the DSP automatically pops the status stack.

To flag that a VIRPT interrupt is pending, the DSP sets the VIPD bit in the SYSTAT register when the external processor writes to the VIRPT register.

Programs passing interprocessor commands must monitor VIPD to check if the DSP can receive a new VIRPT address because:

- If an external processor writes VIRPT while a previous vector is pending, the new VIRPT address replaces the previous pending one.
- If an external processor writes VIRPT while a previous vector is executing, the new VIRPT address does not execute (no new interrupt is triggered).

When returning from a VIRPT interrupt, the DSP clears the VIPD bit. Note that if a DSP writes to its own VIRPT register, the write is ignored.

Timer and Sequencing

The sequencer includes a programmable interval timer, which appears in Figure 3-2 on page 3-5. Bits in the MODE2, TCOUNT, and TPERIOD registers control timer operations. Table A-3 on page A-7 lists the bits in MODE2.

The bits that control the timer are:

- **Timer enable.** MODE2, Bit 5 (TIMEN), directs the DSP to enable (if 1) or disable (if 0) the timer.
- **Timer count.** (TCOUNT) This register contains the decrementing timer count value, counting down the cycles between timer interrupts.
- **Timer period.** (TPERIOD) This register contains the timer period, indicating the number of cycles between timer interrupts.

The TCOUNT register contains the timer counter. The timer decrements the TCOUNT register each clock cycle. When the TCOUNT value reaches zero, the timer generates an interrupt and asserts the TIMEXP output high for four core cycles (when the timer is enabled) as shown in Figure 3-4 on

page 3-33. On the clock cycle after TCOUNT reaches zero, the timer automatically reloads TCOUNT from the TPERIOD register.

The TPERIOD value specifies the frequency of timer interrupts. The number of cycles between interrupts is TPERIOD + 1. The maximum value of TPERIOD is $2^{32} - 1$.

To start and stop the timer, programs use the MODE2 register's TIMEN bit. With the timer disabled (TIMEN=0), the program loads TCOUNT with an initial count value and loads TPERIOD with the number of cycles for the desired interval. Then, the program enables the timer (TIMEN=1) to begin the count.

When a program enables the timer, the timer starts decrementing the TCOUNT register at the end of the next clock cycle. If the timer is subsequently disabled, the timer stops decrementing TCOUNT after the next clock cycle as shown in Figure 3-5.

The timer expired event (TCOUNT decrements to zero) generates two interrupts, TMZHI and TMZLI. For information on latching and masking these interrupts to select timer expired priority, see "Latching Interrupts" on page 3-42.

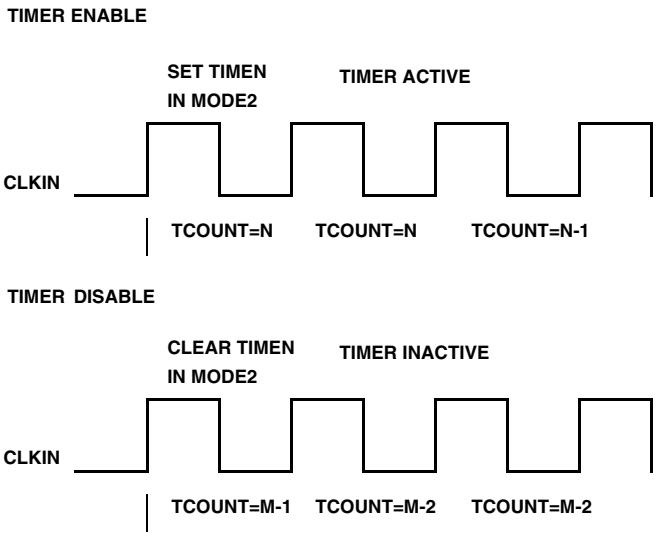


Figure 3-5. Timer Enable and Disable

As with other interrupts, the sequencer needs two cycles to fetch and decode the first instruction of the timer expired service routine, before executing the routine. The pipeline execution for the timer interrupt appears in Table 3-15 on page 3-36.

Programs can read and write the TPERIOD and TCOUNT registers, using universal register transfers. Reading the registers does not effect the timer. Note that an explicit write to TCOUNT takes priority over the sequencer's loading TCOUNT from TPERIOD and the timer's decrementing of TCOUNT. Also note that TCOUNT and TPERIOD are not initialized at reset; programs should initialize these registers before enabling the timer.

Stacks and Sequencing

The sequencer includes a Program Counter (PC) stack, which appears in Table 3-2 on page 3-7. At the start of a subroutine or loop, the sequencer pushes return addresses for subroutines (Call/return instructions) and

top-of-loop addresses for loops (Do/Until) instructions onto the PC stack. The sequencer pops the PC stack during a return from interrupt (RTI), returns from subroutine (RTS), and loop termination.

The PC stack is 30 locations deep. The stack is full when all entries are occupied, is empty when no entries are occupied, and is overflowed if a push occurs when the stack is already full. Bits in the STKYx register indicate the PC stack full and empty states. Table A-5 on page A-13 lists the bits in the STYKx register. The STKYx bits that indicate PC stack status are:

- **PC stack full.** Bit 21, (PCFL), indicates that the PC stack is full (if 1) or not full (if 0)—not a sticky bit, cleared by a Pop.
- **PC stack empty.** Bit 22, (PCEM), indicates that the PC stack is empty (if 1) or not empty (if 0)—not sticky, cleared by a Push.

The PC stack full condition causes a maskable interrupt (SOVFI). This interrupt occurs when the PC stack has 29 locations filled (the almost full state). The PC stack full interrupt occurs when one location is left, because the PC stack full service routine needs that last location for its return address.

The address of the top of the PC stack is available in the PC stack pointer (PCSTKP) register. The value of PCSTKP is zero when the PC stack is empty, is 1...30 when the stack contains data, and is 31 when the stack overflows. This register is a readable and writable register. A write to PCSTKP takes effect after a one-cycle delay. If the PC stack is overflowed, a write to PCSTKP has no effect.

The overflow and full flags provide diagnostic aid only. Programs should not use these flags for runtime recovery from overflow. Note that the status stack, loop stack overflow, and PC stack full conditions trigger a maskable interrupt.

The empty flags can ease stack saves to memory. Programs can monitor the empty flag when saving a stack to memory to determine when the DSP has transferred all values.

Conditional Sequencing

The sequencer supports conditional execution with conditional logic that appears in Figure 3-2 on page 3-5. This logic evaluates conditions for conditional (If) instructions and loop (Do/Until) terminations. The conditions are based on information from the arithmetic status registers (ASTATx and ASTATy), the mode control 1 register (MODE1), the flag inputs and the loop counter. For more information on arithmetic status, see "Using Computational Status" on page 2-7. When in SIMD mode, conditional execution is effected by the arithmetic status of both processing elements. For information on conditional sequencing in SIMD mode, see "SIMD Mode and Sequencing" on page 3-57.

Each condition that the DSP evaluates has an assembler mnemonic. The condition mnemonics for conditional instructions appear in Table 3-19 on page 3-55. For most conditions, the sequencer can test both true and false states. For example, the sequencer can evaluate ALU equal-to-zero (EQ) and ALU not-equal-to-zero (NZ).

To test conditions that do not appear in Table 3-19, a program can use the Test Flag (TF) condition that is generated from a Bit Test Flag (BTF) instruction. The TF flag is set or cleared as a result of a Bit Test or Bit

XOR instruction, which can test the contents of any of the DSP's system registers, including STKYx and STKYy.

Table 3-19. IF Condition and Do/Until Termination Mnemonics

Condition From	Description	True if…	Mnemonic
ALU	ALU = 0	AZ = 1	EQ
	ALU ≠ 0	AZ = 0	NE
	ALU > 0	footnote[1]	GT
	ALU < zero	footnote[2]	LT
	ALU ≥ 0	footnote[3]	GE
	ALU ≤ 0	footnote[4]	LE
	ALU carry	AC = 1	AC
	ALU not carry	AC = 0	NOT AC
	ALU overflow	AV = 1	AV
	ALU not overflow	AV = 0	NOT AV
Multiplier	Multiplier overflow	MV = 1	MV
	Multiplier not overflow	MV= 0	NOT MV
	Multiplier sign	MN = 1	MS
	Multiplier not sign	MN = 0	NOT MS
Shifter	Shifter overflow	SV = 1	SV
	Shifter not overflow	SV = 0	NOT SV
	Shifter zero	SZ = 1	SZ
	Shifter not zero	SZ = 0	NOT SZ
Bit Test	Bit test flag true	BTF = 1	TF
	Bit test flag false	BTF = 0	NOT TF

Table 3-19. IF Condition and Do/Until Termination Mnemonics (Cont'd)

Condition From	Description	True if…	Mnemonic
Flag Input	Flag0 asserted	FI0 = 1	FLAG0_IN
	Flag0 not asserted	FI0 = 0	NOT FLAG0_IN
	Flag1 asserted	FI1 = 1	FLAG1_IN
	Flag1 not asserted	FI1 = 0	NOT FLAG1_IN
	Flag2 asserted	FI2 = 1	FLAG2_IN
	Flag2 not asserted	FI2 = 0	NOT FLAG2_IN
	Flag3 asserted	FI3 = 1	FLAG3_IN
	Flag3 not asserted	FI3 = 0	NOT FLAG3_IN
Mode	Bus master true		BM
	Bus master false		NOT BM
Sequencer	Loop counter expired (Do)	CURLCNTR = 1	LCE
	Loop counter not expired (If)	CURLCNTR ≠ 1	NOT ICE
	Always false (Do)	Always	FOREVER
	Always true (If)	Always	TRUE

1 ALU greater than (GT) is true if: [$\overline{AF}$ and (AN xor (AV and $\overline{ALUSAT}$)) or (AF and AN)] or $\overline{AZ}$ = 0
2 ALU less than (LT) is true if: [$\overline{AF}$ and (AN xor (AV and $\overline{ALUSAT}$)) or (AF and AN and $\overline{AZ}$)] = 1
3 ALU greater equal (GE) is true if: [- and (AN xor (AV and $\overline{ALUSAT}$)) or (AF and AN and $\overline{AZ}$)] = 0
4 ALU lesser or equal (LT) is true if: [$\overline{AF}$ and (AN xor (AV and $\overline{ALUSAT}$)) or (AF and AN)] or $\overline{AZ}$ = 1

The two conditions that do not have complements are LCE/Not LCE (loop counter expired/not expired) and True/Forever. The context of these condition codes determines their interpretation. Programs should use True and Not LCE in conditional (`IF`) instructions. Programs should use Forever and LCE to specify loop (Do/Until) termination. A Do Forever instruction executes a loop indefinitely, until an interrupt or reset intervenes.

There are some restrictions on how programs may use conditions in Do/Until loops. For more information, see "Restrictions On Ending Loops" on page 3-23 and "Restrictions On Short Loops" on page 3-24.



The bus master condition (BM) indicates whether the DSP is the current bus master in a multiprocessor system. To enable testing this condition, a program must clear the MODE1 register's Condition Code Select (CSEL) bits. Otherwise, the bus master condition is always false.

SIMD Mode and Sequencing

The DSP supports a Single-Instruction, Multiple-Data (SIMD) mode. In this mode, both of the DSP's processing elements (PEx and PEy) execute instructions and generate status conditions. For more information on SIMD computations, see "Secondary Processing Element (PEy)" on page 2-36.

Because the two processing elements can generate different outcomes, the sequencers must evaluate conditions from both elements (in SIMD mode) for conditional (If) instructions and loop (Do/Until) terminations. The DSP records status for the PEx element in the ASTATx and STKYx registers. The DSP records status for the PEy element in the ASTATy and STKYy registers. Table A-4 on page A-9 lists the bits in ASTATx and ASTATy, and Table A-5 on page A-13 lists the bits in STKYx and STKYy.

Even though the DSP has dual processing elements, the sequencer does not have dual sets of stacks. There is one PC stack, one loop address stack, and one loop counter stack. The status bits for stacks are in STKYx and are not duplicated in STKYy. In SIMD mode, the status stack stores both ASTATx and ASTATy. A status stack Push or Pop instruction in SIMD mode effects both registers in parallel.

While in SIMD mode, the sequencer evaluates conditions from both PE's for conditional (If) and loop (Do/Until) instructions. Table 3-20 summa-

rizes how the sequencer resolves each conditional test when SIMD mode is enabled.

Table 3-20. Conditional Execution Summary

Conditional Operation	Conditional Outcome Depends On …
Compute Operations	Executes in each PE independently depending on condition test in each PE
Branches and Loops	Executes in sequencer depending on And'ing condition test on both PE's.
Data Moves (from complementary pair[1] to complementary pair, including X<->Y swap)	Executes move in each PE (and/or memory) independently depending on condition test in each PE
Data Moves (from uncomplemented universal register[2] to complementary pair[1])	Executes move in each PE (and/or memory) independently depending on condition test in each PE; the same uncomplemented universal is source for each move
Data Moves (from complementary pair[1] to uncomplemented register[2])	Executes explicit move to uncomplemented universal register depending on condition test in PEx only; no implicit move occurs
DAG Operations	Executes modify[3] in DAG depending on Or'ing condition test on both PE's

1 Complementary pairs are registers with SIMD complements, include PEx/y data registers and USTAT1/2, USTAT3/4, ASTATx/y, STKYx/y, and PX1/2 universal registers. This also includes internal memory for conditional execution.
2 Uncomplemented registers are universal registers that do not have SIMD complements.
3 Post-modify operations follow this rule, but pre-modify operations always occur despite outcome

Conditional Compute Operations

While in SIMD mode, a conditional compute operation may execute on both PE's, either one PE, or neither PE dependent on the outcome of the status flag test. Flag testing is independently performed on each PE.

Conditional Branches and Loops

The DSP executes a conditional branch (Jump or Call/return) or loop (Do/Until) based on the result of And'ing the condition tests on both PE's. A conditional branch or loop in SIMD mode occurs only when the condition is true in PEx and PEy.

Using complementary conditions (for example EQ and NE), programs can produce an OR'ing of the condition tests for branches and loops in SIMD mode. A conditional branch or loop that uses this technique should consist of a series of conditional compute operations. These conditional computes generate NOPs on the PE where a branch or loop does not execute. For more information on programming in SIMD mode, see the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Conditional Data Moves

The execution of a conditional (If) data move (register-to-register and register-to/from-memory) instruction depends on three factors:

- The explicit data move depends on the evaluation of the conditional test in the PEx processing element
- The implicit data move depends on the evaluation of the conditional test in the PEy processing element
- Both moves depend on the types of registers used in the move

There are four cases for SIMD conditional data moves.

Case 1: Complementary Register Pair Data Move

In this case data moves from a complementary register pair to a complementary register pair. The DSP executes the explicit move depending on the evaluation of the conditional test in the PEx processing element and

the implicit move depending on the evaluation of the conditional test in the PEy processing element.

Example: Register to Memory Move – PEx Explicit Register

```
IF EQ DM(I0,M0) = R2;
```

For this instruction the DSP is operating in SIMD mode, a register in the PEx data register file is the explicit register and `I0` is pointing to an even address in internal memory. Indirect addressing is shown in the instructions shown in this example. However, the same results occur using direct addressing. The data movement resulting from the evaluation of the conditional test in the PEx and PEy processing elements is shown in Table 3-21.

Table 3-21. Register to Memory Moves – Complementary Pairs

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	NO data move occurs	NO data move occurs
0	1	NO data move occurs from r2 to location I0	s2 transfers to location (I0+1)
1	0	r2 transfers to location I0	NO data move occurs from s2 to location (I0+1)
1	1	r2 transfers to location I0	s2 transfers to location (I0+1)

Example: Register to Memory Move – PEy Explicit Register

```
IF EQ DM(I0,M0) = S2;
```

For this instruction the DSP is operating in SIMD mode, a register in the PEy data register file is the explicit register and `I0` is pointing to an even address in internal memory. The data movement resulting from the evalu-

ation of the conditional test in the PEx and PEy processing elements is shown in Table 3-22.

Table 3-22. Register to Register Moves – Complementary Pairs

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	NO data move occurs	NO data move occurs
0	1	NO data move occurs from s2 to location I0	r2 transfers to location I0+1
1	0	s2 transfers to location I0	NO data move occurs from r2 to location I0+1
1	1	s2 transfers to location I0	r2 transfers to location I0+1

Examples: Register to Register Move Instructions

```
IF EQ R8 = R2;
IF EQ PX1 = R2;
IF EQ USTAT1 = R2;
```

For these instruction the DSP is operating in SIMD mode and registers in the PEx data register file are used as the explicit registers. The data move-

ment resulting from the evaluation of the conditional test in the PEx and PEy processing elements is shown in Table 3-23.

Table 3-23. Register to Register Moves – Complementary Pairs

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	NO data move occurs	NO data move occurs
0	1	NO data move to registers r9,px1,ustat1 occurs	s2 transfers to registers s9,px2 and ustat2
1	0	r2 transfers to registers r9,px1 and ustat1	NO data move to s9, px2, or ustat2 occurs
1	1	r2 transfers to registers r9,px1, and ustat1	s2 transfers to registers s9,px2,and ustat2

Examples: Register to Register Move Instructions

```
IF EQ R8 = S2;
IF EQ PX1 = S2;
IF EQ USTAT1 = S2;
```

For these instructions the DSP is operating in SIMD mode and registers in the PEy data register file are used as explicit registers. The data movement resulting from the evaluation of the conditional test in the PEx and PEy processing elements is shown in Table 3-24.

Table 3-24. Register to Register Moves – Complementary Register Pairs

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	NO data move occurs	NO data move occurs
0	1	NO data move to registers s9,px and ustat1 occurs	r2 transfers to registers s9,px2, and ustat2
1	0	s2 transfers to registers r9,px1 and ustat1	NO data move to registers s9,px2, and ustat2 occurs
1	1	s2 transfers to registers r9,px1, and ustat1	r2 transfers to registers s9,px2, and ustat2

Case 2: Uncomplemented to Complementary Register Move

In this case data moves from an uncomplemented register (Ureg without a SIMD complement) to a complementary register pair. The DSP executes the explicit move depending on the evaluation of the conditional test in the PEx processing element. The DSP executes the implicit move depending on the evaluation of the conditional test in the PEy processing element. In each processing element where the move occurs, the content of the source register is duplicated in destination.

Example: Register to Register Move

```
IF EQ R1 = PX;
```



While `PX1` and `PX2` are complementary registers, the combined `PX` register has no complementary register. For more information, see “Internal Data Bus Exchange” on page 5-7.

For this instruction the DSP is operating in SIMD mode. The data movement resulting from the evaluation of the conditional test in the PEx and PEy processing elements is shown in Table 3-25.

Table 3-25. Complementary to Uncomplemented Register Move

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	r1 remains unchanged	s1 remains unchanged
0	1	r1 remains unchanged	s1 gets px value
1	0	r1 gets px value	s1 remains unchanged
1	1	r1 gets px value	s1 gets px value

Case 3: Complementary Register => Uncomplimentary Register

In this case data moves from a complementary register pair to an uncomplemented register. The DSP executes the explicit move to the uncomplemented universal register, depending on the condition test in the PEx processing element only. The DSP does not perform an implicit move.

Example: Register to Register Move

```
IF EQ PX = R1;
```

For this instruction the DSP is operating in SIMD mode. The data movement resulting from the evaluation of the conditional test in the PEx and PEy processing elements is shown in Table 3-26.

Table 3-26. Complementary to Uncomplemented Move

Condition in PEx	Condition in PEy	Result	
AZx	AZy	Explicit	Implicit
0	0	r1 remains unchanged	no implicit move s1 remains unchanged
0	1	r1 remains unchanged	no implicit move s1 remains unchanged
1	0	r1 40-bit explicit move to px	no implicit move s1 remains unchanged
1	1	r1 40-bit explicit move to px	no implicit move s1 remains unchanged

For more details on PX register transfers, refer to "Internal Data Bus Exchange" on page 5-7.

Case 4: Data Move Involves External Memory or IOP Memory Space

Conditional data moves from a complementary register pair to an uncomplemented register with an access to external memory space or IOP memory space. This results in unexpected behavior and should not be used.

Example: Register to Memory Move

```
IF EQ DM(I0,M0) = R2;
IF EQ DM(I0,M0) = S2;
```

For these instruction the DSP is operating in SIMD mode and the explicit register is either a PEx register or PEy register. I0 points to either external memory space or IOP memory space.

Indirect addressing is shown in the instructions shown in this example. However, the same results occur using direct addressing.

Conditional DAG Operations

Conditional post-modify DAG operations update the DAG register based on OR'ing of the condition tests on both processing elements. Actual data movement involved in a conditional DAG operation is based on independent evaluation of condition tests in PEx and PEy. Only the post modify update is based on the OR'ing of the these conditional tests.

Conditional pre-modify DAG operations behave differently. The DAGs always pre-modify an index, independent of the outcome of the condition tests on each processing element.

4 DATA ADDRESS GENERATORS

This chapter describes Data Address Generators (DAGs).

Overview

The DSP's Data Address Generators (DAGs) generate addresses for data moves to and from Data Memory (DM) and Program Memory (PM). By generating addresses, the DAGs let programs refer to addresses indirectly, using a DAG register instead of an absolute address. The DAGs architecture, which appears in Figure 4-1, supports several functions that minimize overhead in data access routines. These functions include:

- **Supply address and post-modify**—provides an address during a data move and auto-increments the stored address for the next move.
- **Supply pre-modified address**—provides a modified address during a data move without incrementing the stored address.
- **Modify address**—increments the stored address without performing a data move.
- **Bit-reverse address**—provides a bit-reversed address during a data move without reversing the stored address.
- **Broadcast data moves**—performs dual data moves to complementary registers in each processing element to support SIMD mode.

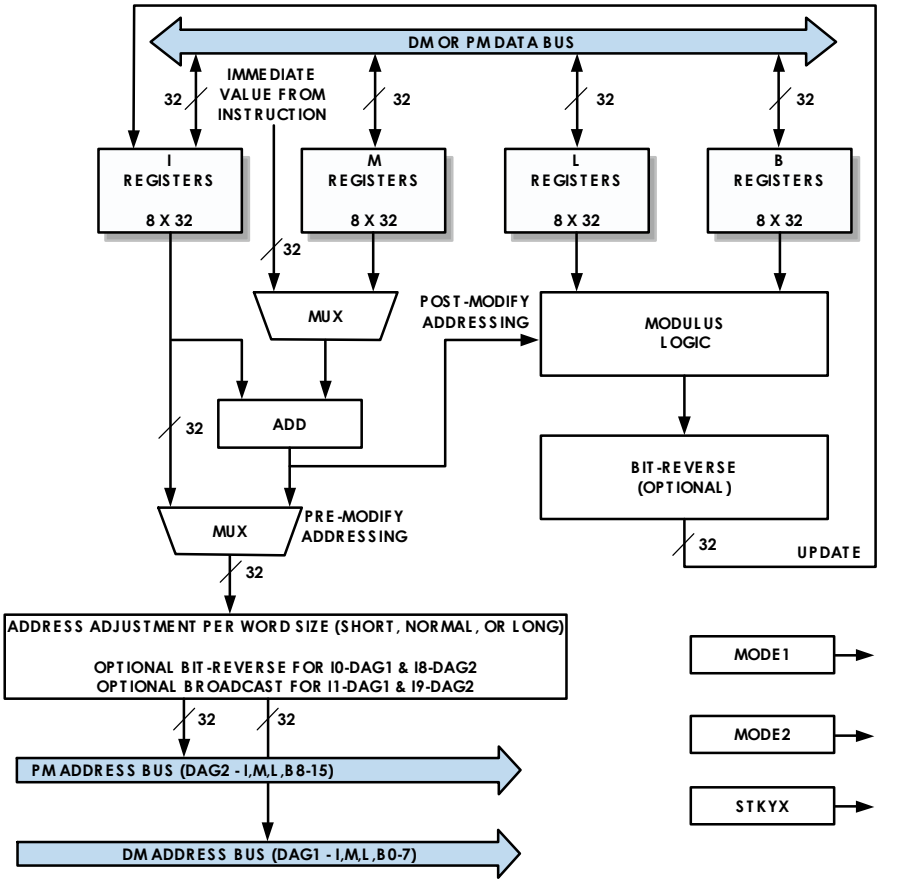


Figure 4-1. Data Address Generator (DAG) Block Diagram

As shown in Figure 4-1, each DAG has four types of registers. These registers hold the values that the DAG uses for generating addresses. The four types of registers are:

- **Index registers (I0-I7 for DAG1 and I8-I15 for DAG2).** An index register holds an address and acts as a pointer to memory. For example, the DAG interprets `dm(I0,0)` and `pm(I8,0)` syntax in an instruction as addresses.
- **Modify registers (M0-M7 for DAG1 and M8-M15 for DAG2).** A modify register provides the increment or step size by which an index register is pre- or post-modified during a register move. For example, the `dm(I0, M1)` instruction directs the DAG to output the address in register `I0` then modify the contents of `I0` using the `M1` register.
- **Length and Base registers (L0-L7 and B0-B7 for DAG1 and L8-L15 and B8-B15 for DAG2).** Length and base registers set up the range of addresses and the starting address for a circular buffer. For more information on circular buffers, see "Addressing Circular Buffers" on page 4-12.

Setting DAG Modes

The `MODE1` register controls the operating mode of the DAGs. Table A-2 on page A-3 lists all the bits in `MODE1`. The following bits in `MODE1` control Data Address Generator modes:

- **Circular buffering enable.** Bit 24 (`CBUFEN`) enables circular buffering (if 1) or disables circular buffering (if 0).
- **Broadcast register loading enable, DAG1-I1.** Bit 23 (`BDCST1`) enables register broadcast loads to complementary registers from I1 indexed moves (if 1) or disables broadcast loads (if 0).

- **Broadcast register loading enable, DAG2-I9.** Bit 22 (BDCST9) enables register broadcast loads to complementary registers from I9 indexed moves (if 1) or disables broadcast loads (if 0)
- **SIMD mode enable.** Bit 21 (PEYEN) enables computations in PEy—SIMD mode—(if 1) or disables PEy—SISD mode—(if 0). For more information on SIMD mode, see "Secondary Processing Element (PEy)" on page 2-36.
- **Secondary registers for DAG2 lo, I,M,L,B8-11.** Bit 6 (SRD2L)
  Secondary registers for DAG2 hi, I,M,L,B12-15. Bit 5 (SRD2H)
  Secondary registers for DAG1 lo, I,M,L,B0-3. Bit 4 (SRD1L)
  Secondary registers for DAG1 hi, I,M,L,B4-7. Bit 3 (SRD1H)
  These bits select the corresponding secondary register set (if 1) or select the corresponding primary register set—the set that is available at reset—(if 0).
- **Bit-reverse addressing enable, DAG1-I0.** Bit 1 (BR0) enables bit-reversed addressing on I0 indexed moves (if 1) or disables bit-reversed addressing (if 0).
- **Bit-reverse addressing enable, DAG2-I8.** Bit 0 (BR8) enables bit-reversed addressing on I8 indexed moves (if 1) or disables bit-reversed addressing (if 0).

Circular Buffering Mode

The CBUFEN bit in the MODE1 register enables circular buffering—a mode in which the DAG supplies addresses ranging within a constrained buffer length (set with an L register), starting at a base address (set with a B regis-

ter), and incrementing the addresses on each access by a modify value (set with an M register).



On previous SHARC DSP's (ADSP-2106x DSPs), circular buffering is always enabled. For code compatibility, programs ported to the ADSP-21160 DSP should include the instruction:

```
Bit Set Mode1 CBUFEN;
```

This instruction enables circular buffering. For more information on setting up and using circular buffers, see "Addressing Circular Buffers" on page 4-12. When using circular buffers, the DAGs can generate an interrupt on buffer overflow (wrap around). For more information, see "Using DAG Status" on page 4-9.

Broadcast Loading Mode

The `BDCST1` and `BDCST9` bits in the `MODE1` register enable broadcast loading mode—multiple register loads from a single load command. When the `BDCST1` bit is set (1), the DAG performs a dual data register load on instructions that use the `I1` register for the address. The DAG loads both the named register (explicit register) in one processing element and loads that register's complementary register (implicit register) in the other processing element. The `BDCST9` bit in the `MODE1` register enables this feature for the `I9` register.

Enabling either DAG1 or DAG2 register load broadcasting has no effect on register stores or loads to universal registers other than the register file data registers. "Dual Processing Element Register Load Broadcasts" on page 4-6 demonstrates the effects of a register load operation on both pro-

cessing elements with register load broadcasting enabled. In Table 4-1 on page 4-6, note that Rx and Sx are complementary data registers.

Table 4-1. Dual Processing Element Register Load Broadcasts

[1]	**Example**
Instruction syntax	`Rx = DM(I1,Ma);` {Syntax #1} `Rx = PM(I9,Mb);` {Syntax #2} `Rx = DM(I1,Ma), Rx = PM(I9,Mb);` {Syntax #3}
PEx explicit operations	`Rx = DM(I1,Ma);` {Explicit #1} `Rx = PM(I9,Mb);` {Explicit #2} `Rx = DM(I1,Ma), Rx = PM(I9,Mb);` {Explicit #3}
PEy implicit operations	`Sx = DM(I1,Ma);` {Implicit #1} `Sx = PM(I9,Mb);` {Implicit #2} `Sx = DM(I1,Ma), Sx = PM(I9,Mb);` {Implicit #3}

1 The letters “a” and “b” (as in Ma or Mb) indicate numbers for modify registers in DAG1 and DAG2. The letter “a”, which indicates a DAG1 register, can be replaced with 0 through 7. The letter “b” indicates a DAG2 register and can be replaced with 8 through 15.



The `PEYEN` bit (SISD/SIMD mode select) does not influence broadcast operations. Broadcast loading is particularly useful in SIMD applications where the algorithm needs identical data loaded into each processing element. For more information on SIMD mode (in particular, a list of complementary data registers), see “Secondary Processing Element (PEy)” on page 2-36.

Alternate (Secondary) DAG Registers

Each DAG has an alternate register set. To facilitate fast context switching, the DSP includes alternate register sets for data, results, and data address generator registers. Bits in the `MODE1` register control when alternate registers become accessible. While inaccessible, the contents of alternate registers are not effected by DSP operations. Note that there is a one cycle latency between writing to `MODE1` and being able to access an alternate register set. The alternate register sets for the DAGs are described

in this section. For more information on alternate data and results registers, see "Alternate (Secondary) Data Registers" on page 2-31.

Bits in the MODE1 register can activate alternate register sets within the DAGs: the lower half of DAG1 (I,M,L,B0-3), the upper half of DAG1 (I,M,L,B4-7), the lower half of DAG2 (I,M,L,B8-11), and the upper half of DAG2 (I,M,L,B12-15). Figure 4-2 shows the DAG's primary and alternate register sets.

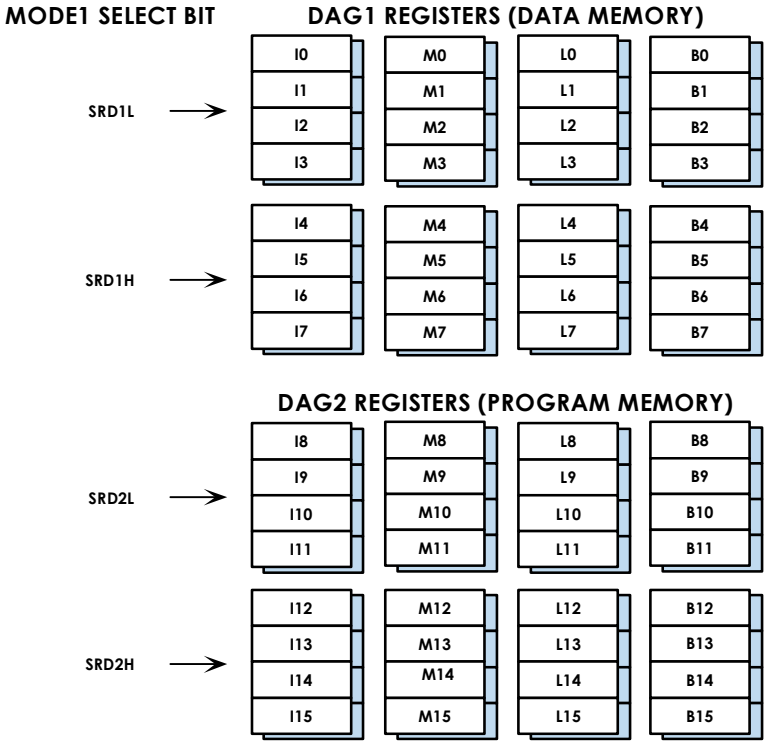


Figure 4-2. DAG Primary and Alternate Registers

To share data between contexts, a program places the data to be shared in one half of either the current DAG's registers or the other DAG's registers and activates the alternate register set of the other half. The following

example demonstrates how code should handle the one cycle of latency from the instruction setting the bit in MODE1 to when the alternate registers may be accessed:

```
BIT SET MODE1 SRD1L;/* activate alternate dag1 lo regs */
NOP;/* wait for access to alternates */
R0=dm(i0,m1);
```

Bit-Reverse Addressing Mode

The BR0 and BR8 bits in the MODE1 register enable bit-reverse addressing mode—outputting addresses in reverse bit order. When BR0 is set (1), DAG1 bit-reverses 32-bit addresses output from I0. When BR8 is set (1), DAG2 bit-reverses 32-bit addresses output from I8. The DAGs only bit-reverse the address output from I0 or I8; the contents of these registers are not reversed. Bit-reverse addressing mode effects both pre-modify and post-modify operations. The following example demonstrates how bit-reverse mode effects address output:

```
Bit Set Mode1 BR0; /* enables bit-rev. addressing for DAG1 */
I0=0x8a000; /* loads I0 with the bit reverse of the */
               buffer's base address, DM(0x51000) */
M0=0x4000000; /* loads M0 with value for post-modify */
R1=DM(I0,M0); /* loads r1 with contents of DM address   */
              /* DM(0x51000), which is the bit-reverse of */
              /* 0x8a000, then post modifies I0 for the next */
              /* access with (0x8a000 + 0x4000000)=0x408a000, */
              /* which is the bit-reverse of DM(0x51020) */
```

In addition to bit-reverse addressing mode, the DSP supports a bit-reverse instruction (Bitrev). This instruction bit-reverses the contents of the selected register. For more information on the Bitrev instruction, see "Modifying DAG Registers" on page 4-18 or the *ADSP-21160 SHARC DSP Instruction Set Reference*.

Using DAG Status

As described in "Addressing Circular Buffers" on page 4-12, the DAGs can provide addressing for a constrained range of addresses, repeatedly cycling through this data (or buffer). A buffer overflow (or wrap around) occurs each time the DAG circles past the buffer's base address.

The DAGs can provide buffer overflow information when executing circular buffer addressing for `I7` or `I15`. When a buffer overflow occurs (a circular buffering operation increments the I register past the end of the buffer), the appropriate DAG updates a buffer overflow flag in a sticky status (`STKYx`) register. A buffer overflow can also generate a maskable interrupt. Two ways to use buffer overflows from circular buffering are:

- **Interrupts.** Enable interrupts and use an interrupt service routine to handle the overflow condition immediately. This method is appropriate if it is important to handle all overflows as they occur; for example in a "ping-pong" or swap I/O buffer pointers routine.
- **STKYx registers.** Use the Bit Tst instruction to examine overflow flags in the `STKY` register after a series of operations. If an overflow flag is set, the buffer has overflowed—wrapped around—at least once. This method is useful when overflow handling is not critical.

DAG Operations

The DSP's DAGs perform several types of operations to generate data addresses. As shown in Figure 4-1 on page 4-2, the DAG registers and the

MODE1, MODE2, and STKYx registers all contribute to DAG operations. The following sections provide details on DAG operations:

- "Addressing With DAGs" on page 4-10
- "Addressing Circular Buffers" on page 4-12
- "Modifying DAG Registers" on page 4-18

An important item to note from Figure 4-1 on page 4-2 is that the DAG automatically adjusts the output address per the word size of the address location (short word, normal word, or long word). This address adjustment lets internal memory use the address directly. For details on these address adjustments, see "Access Word Size" on page 5-40.



SISD/SIMD mode, access word size, and data location (internal/external) all influence data access operations. "Data Access Options" on page 5-45

Addressing With DAGs

The DAGs support two types of modified addressing—generating an address that is incremented by a value or a register. In pre-modify addressing, the DAG adds an offset (modifier), either an M register or an immediate value, to an I register and outputs the resulting address. Pre-modify addressing does not change (or update) the I register. The other type of modified addressing is post-modify addressing. In post-modify addressing, the DAG outputs the I register value unchanged then adds

an M register or immediate value, updating the I register value. Figure 4-3 compares pre- and post-modify addressing.

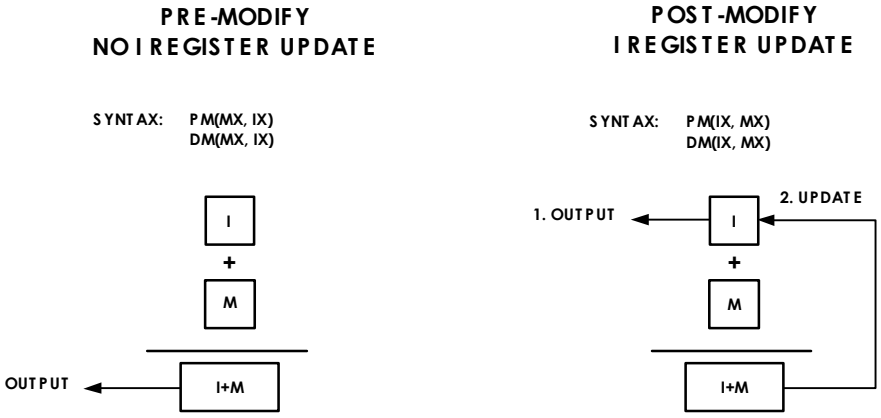


Figure 4-3. Data Address Generator (DAG) Block Diagram

The difference between pre-modify and post-modify instructions in the DSP's assembly syntax is the position of the index and modifier in the instruction. If the I register comes before the modifier, the instruction is a post-modify operation. If the modifier comes before the I register, the instruction is a pre-modify without update operation. The following instruction accesses the program memory location indicated by the value in `I15` and writes the value `I15` + `M12` to the `I15` register:

```
R6 = PM(I15,M12); /* Post-modify addressing with update */
```

By comparison, the following instruction accesses the program memory location indicated by the value `I15` + `M12` and does not change the value in `I15`:

```
R6 = PM(M12,I15); /* Pre-modify addressing without update */
```

Modify (M) registers can work with any index (I) register in the same DAG (`DAG1` or `DAG2`). For a list of I and M registers and their DAGs, see Figure 4-2 on page 4-7.

Instructions can use a number (immediate value), instead of an M register, as the modifier. The size of an immediate value that can modify an I register depends on the instruction type. For all single data access operations, modify immediate values can be up to 32 bits wide. Instructions that combine DAG addressing with computations limit the size of the modify immediate value. In these instructions (multifunction computations), the modify immediate values can be up to 6 bits wide. The following example instruction accepts up to 32-bit modifiers:

```
R1=DM(0x40000000,I1);    /* DM address = I1+0x4000 0000 */
```

The following example instruction accepts up to 6-bit modifiers:

```
F6=F1+F2,PM(I8,0x0B)=ASTAT; /* PM address = I8, I8=I8+0x0B */
```

Note that pre-modify addressing operations must not change the memory space of the address. For example, pre-modifying an address in the DSP's internal memory space should not generate an address in external memory space. For more information, see "Access Word Size" on page 5-40.

Addressing Circular Buffers

The DAGs support addressing circular buffers—a range of addresses containing data that the DAG steps through repeatedly, "wrapping around" to repeat stepping through the range of addresses in a circular pattern. To address a circular buffer, the DAG steps the index pointer (I register) through the buffer, post-modifying and updating the index on each access with a positive or negative modify value (M register or immediate value). If the index pointer falls outside the buffer, the DAG subtracts or adds the length of the buffer from or to the value, wrapping the index pointer back to the start of the buffer. The DAG's support for circular buffer addressing appears in Figure 4-1 on page 4-2, and an example of circular buffer addressing appears in Figure 4-1.

The starting address that the DAG wraps around is called the buffer's base address (B register). There are no restrictions on the value of the base address for a circular buffer.



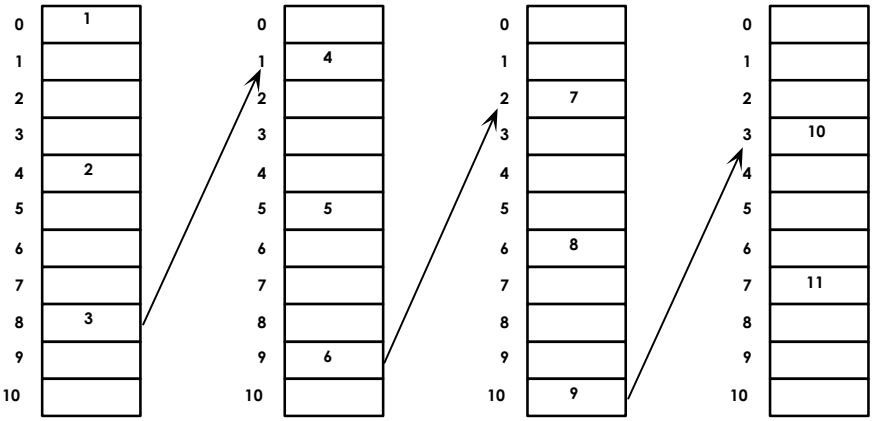
Circular buffering may only use post-modify addressing. The DAG's architecture, as shown in Figure 4-1 on page 4-2, cannot support pre-modify addressing for circular buffering, because circular buffering requires that the index be updated on each access.

It is important to note that the DAGs do not detect memory map overflow or underflow. If the address post-modify produces I+M > 0xffffffff or I–M < 0, circular buffering may not function correctly. Also, the length of a circular buffer should not let the buffer straddle the top of the memory map. For more information on the DSP's memory map, see "ADSP-21160 DSP Memory Map" on page 5-12.

As shown in Figure 4-4 on page 4-14, programs use the following steps to set up a circular buffer:

```
THE FOLLOWING SYNTAX SETS UP AND ACCESSES A CIRCULAR BUFFER WITH:
        LENGTH = 11
        BASE ADDRESS = 0X55000
        MODIFIER = 4

        BIT SET MODE1 CBUFEN;          /* ENABLES CIRCULAR BUFFER ADDRESSING; JUST ONCE IN PROGRAM  */
        B0 = 0X55000;                  /* LOADS B0 AND L0 REGISTERS WITH BASE ADDRESS */
        L0 = 0XB;                      /* LOADS L0 REGISTER WITH LENGTH OF BUFFER */
        M1 = 0X4;                      /* LOADS M1 WITH MODIFIER OR STEP SIZE */
        LCNTR = 11, DO MY_CIR_BUFFER UNTIL LCE;   /* SETS UP A LOOP CONTAINING BUFFER ACCESSES */
        R0 = DM(I0,M1);                /* AN ACCESS WITHIN THE BUFFER USES POST MODIFY ADDRESSING */
        ...                            /* OTHER INSTRUCTIONS IN THE MY_CIR_BUFFER LOOP */
        MY_CIR_BUFFER: NOP;            /* END OF MY_CIR_BUFFER LOOP */
```



THE COLUMNS ABOVE SHOW THE SEQUENCE IN ORDER OF LOCATIONS ACCESSED IN ONE PASS.
NOTE THAT "0" ABOVE IS ADDRESS DM(0X55000). THE SEQUENCE REPEATS ON SUBSEQUENT PASSES.

Figure 4-4. Data Address Generator (DAG) Block Diagram

1. Enable circular buffering (`Bit Set Mode1 CBUFEN;`). This operation is only needed once in a program.
2. Load the buffer's base address into the B register. This operation automatically loads the corresponding I register.

3. Load the buffer's length into the corresponding L register. For example, L0 corresponds to B0.
4. Load the modify value (step size) into an M register in the corresponding DAG. For example, M0 through M7 correspond to B0. Alternatively, the program can use an immediate value for the modifier.

After this set up, the DAGs use the modulus logic in Figure 4-1 on page 4-2 to process circular buffer addressing.

On the ADSP-21160 DSP, programs enable circular buffering by setting the CBUFEN bit in the MODE1 register. This bit has a corresponding mask bit in the MMASK register. Setting the corresponding MMASK bit causes the CBUFEN bit to be cleared following a push status instruction (Push Sts), the execution of an external interrupt, timer interrupt, or vectored interrupt. This feature lets programs disable circular buffering while in an interrupt service routine that does not use circular buffering. By disabling circular buffering, the routine does not need to save and restore the DAG's B and L registers.

Clearing the CBUFEN bit disables circular buffering for all data load and store operations. The DAGs perform normal post-modify load and store accesses instead, ignoring the B and L register values. Note that a write to a B register modifies the corresponding I register, independent of the state of the CBUFEN bit. The Modify instruction executes independent of the state of the CBUFEN bit. The Modify instruction always performs circular buffer modify of the index registers if the corresponding B and L registers are set up, independent of the state of the CBUFEN bit.



On previous SHARC DSP's (ADSP-2106x family), circular buffering is always enabled. For code compatibility, programs ported to the ADSP-21160 DSP should enable circular buffering (CBUFEN=1).

On the first post-modify access to the buffer, the DAG outputs the I register value on the address bus then modifies the address by adding the

modify value. If the updated index value is within the buffer length, the DAG writes the value to the I register. If the updated value is outside the buffer length, the DAG subtracts (positive) or adds (negative) the L register value before writing the updated index value to the I register. In equation form, these post-modify and wrap around operations work as follows:

- If M is positive:
    - $I_{new} = I_{old} + M$ if $I_{old} + M <$ Buffer base + length (end of buffer)
    - $I_{new} = I_{old} + M - L$ if $I_{old} + M \geq$ Buffer base + length (end of buffer)
- If M is negative:
    - $I_{new} = I_{old} + M$ if $I_{old} + M \geq$ Buffer base (start of buffer)
    - $I_{new} = I_{old} + M + L$ if $I_{old} + M <$ Buffer base (start of buffer)

The DAGs use all four types of DAG registers for addressing circular buffers. These registers operate as follows for circular buffering:

The index (I) register contains the value that the DAG outputs on the address bus.

- The modify (M) register contains the post-modify amount (positive or negative) that the DAG adds to the I register at the end of each memory access. The M register can be any M register in the same DAG as the I register and does not have to have the same number. The modify value also can be an immediate value instead

of an M register. The size of the modify value, whether from an M register or immediate, must be less than the length (L register) of the circular buffer.

- The length (L) register sets the size of the circular buffer and the address range that the DAG circulates the I register through. L must be positive and cannot have a value greater than $2^{31} - 1$. If an L register's value is zero, its circular buffer operation is disabled.
- The base (B) register, or the B register plus the L register, is the value that the DAG compares the modified I value with after each access. When the B register is loaded, the corresponding I register is simultaneously loaded with the same value. When I is loaded, B is not changed. Programs can read the B and I registers independently.

There is one set of registers (I7 and I15) in each DAG that can generate an interrupt on circular buffer overflow (address wraparound). For more information, see "Using DAG Status" on page 4-9.

When a program needs to use I7 or I15 without circular buffering and the DSP has the circular buffer overflow interrupts unmasked, the program should disable the generation of these interrupts by setting the B7/B15 and L7/L15 registers to values that prevent the interrupts from occurring. If I7 were accessing the address range 0x1000–0x2000, the program could set B7=0x0000 and L7=0xFFFF. Because the DSP generates the circular buffer interrupt based on the wrap around equations on page 4-15, setting the L register to zero does not necessarily achieve the desired results. If the program is using either of the circular buffer overflow interrupts, it should avoid using the corresponding I register(s) (I7 or I15) where interrupt branching is not needed.

In the case of circular buffer overflow interrupts, if CBUFEN = 1 and register L7 = 0 (or L15 = 0), the CB7I (or CB15I) interrupt occurs at every change of I7 (or I15) after the index register (I7 or I15) crosses the base register (B7

or B15) value. This wrap-around behavior is independent of the context of the DAG registers, both primary and alternate.



When a Long word access, SIMD access, or Normal word access (with LW option) crosses the end of the circular buffer, the DSP completes the access before responding to the end of buffer condition.

Modifying DAG Registers

The DAGs support two operations that modify an address value in an index register without outputting an address. These two operations, address bit-reversal and address modify, are useful for bit-reverse addressing and maintaining pointers.

The Modify instruction modifies addresses in any DAG index register (I0-I15) without accessing memory. If the I register's corresponding B and L registers are set up for circular buffering, a Modify instruction performs the specified buffer wrap around (if needed). The syntax for Modify is similar to post-modify addressing (index, then modifier). Modify accepts either a 32-bit immediate values or an M register as the modifier. The following example adds 4 to I1 and updates I1 with the new value:

```
Modify(I1,4);
```

The Bitrev instruction modifies and bit-reverses addresses in any DAG index register (I0-I15) without accessing memory. This instruction is independent of the bit-reverse mode. The Bitrev instruction adds a 32-bit immediate value to a DAG index register, bit-reverses the result, and writes the result back to the same index register. The following example adds 4 to I1, bit-reverses the result, and updates I1 with the new value:

```
Bitrev(I1,4);
```

Addressing in SISD and SIMD Modes

Single-Instruction, Multiple-Data (SIMD) mode (PEYEN bit=1) does not change the addressing operations in the DAGs, but it does change the amount of data that moves during each access. The DAGs put the same addresses on the address buses in SIMD and SISD modes. In SIMD mode, the DSP's memory and processing elements get data from the locations named (explicit) in the instruction syntax and complementary (implicit) locations. For more information on data moves between registers, see "Secondary Processing Element (PEy)" on page 2-36. For more information on data accesses and memory, see "Data Access Options" on page 5-45.

DAGs, Registers, and Memory

DAG registers are part of the DSP's universal register set. Programs may load the DAG registers from memory, from another universal register, or with an immediate value. Programs may store DAG registers' contents to memory or to another universal register.

The DAG's registers support the bidirectional register-to-register transfers that are described in "SIMD (Computational) Operations" on page 2-40. When the DAG register is a source of the transfer, the destination can be a register file data register. This transfer results in the contents of the single source register being duplicated in complementary data registers in each processing element.

Programs should use care in the case where the DAG register is a destination of a transfer from a register file data register source. Programs should use a conditional operation to select either one processing element or neither as the source. Having both processing elements contribute a source value results in the PEx element's write having precedence over the PEy element's write.

In the case where a DAG register is both source and destination, the data move operation executes the same as it would if SIMD mode were disabled (PEYEN cleared).

DAG Register-to-Bus Alignment

There are three word alignment cases for DAG registers and PM or DM data buses: Normal word, Extended-precision Normal word, and Long word.

The DAGs align normal word (32-bit) addressed transfers to the low order bits of the buses. These transfers between memory and 32-bit DAG1or DAG2 registers use the 64-bit DM and PM data buses. Figure 4-5 illustrates these transfers.

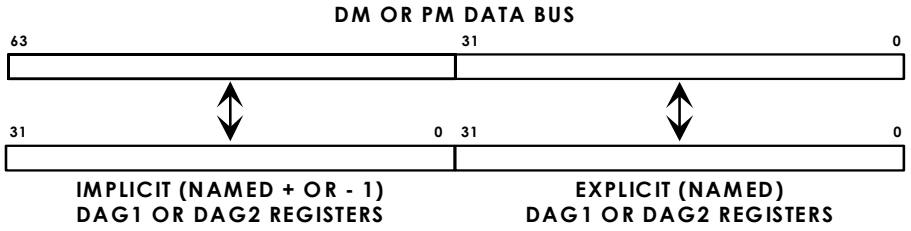


Figure 4-5. Normal Word (32-bit) DAG Register Memory Transfers

The DAGs align extended-precision normal word (40-bit) addressed transfers or register-to-register transfers to bits 39-8 of the buses. These transfers between a 40-bit data register and 32-bit DAG1 or DAG2 regis-

ters use the 64-bit DM and PM data buses. Figure 4-6 illustrates these transfers.

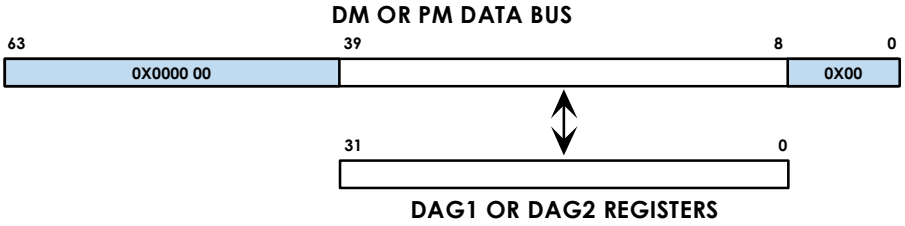


Figure 4-6. DAG Register to Data Register Transfers

Long word (64-bit) addressed transfers between memory and 32-bit DAG1 or DAG2 registers target double DAG registers and use the 64-bit DM and PM data buses. Figure 4-7 illustrates how the bus works in these transfers.

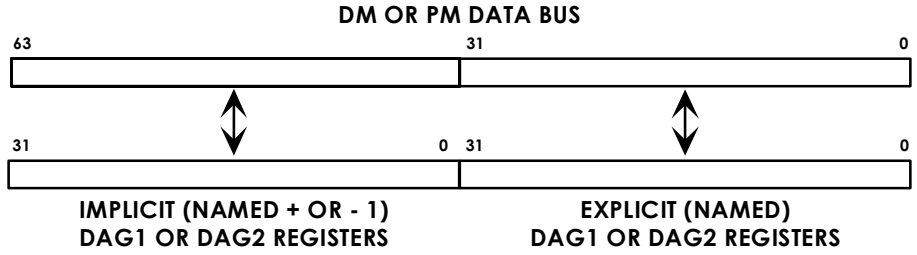


Figure 4-7. Long Word DAG Register to Data Register Transfers

If the Long word transfer specifies an even-numbered DAG register (e.g., `I0` or `I2`), then the even numbered register value transfers on the lower half of the 64-bit bus, and the even numbered register + 1 value transfers on the upper half (bits 63-32) of the bus.

If the Long word transfer specifies an odd numbered DAG register (e.g., `I1`, or `B3`), the odd numbered register value transfers on the lower half of the 64-bit bus, and the odd numbered register - 1 value (`I0` or `B2` in this example) transfers on the upper half (bits 63-32) of the bus.

In both the even- and odd-numbered cases, the explicitly specified DAG register sources or sinks bits 31-0 of the Long word addressed memory.

DAG Register Transfer Restrictions

The two types of transfer restrictions are hold-off conditions and illegal conditions that the DSP does not detect.

For certain instruction sequences involving transfers to and from DAG registers, an extra (Nop) cycle is automatically inserted by the processor. When an instruction that loads a DAG register is followed by an instruction that uses the same DAG register for data addressing, modify instructions, or indirect jumps, the DSP inserts an extra (Nop) cycle between the two instructions. This hold-off happens because the same bus is needed by both operations in the same cycle. So, the second operation must be delayed. The following case causes a delay because it exhibits a write/read dependency in which `I0` is written in one cycle. The results of that register write are not available to a register read for one cycle. Note that if either instruction had specified `I1`, the stall would still occur, because the DSP's DAG register transfers can occur in pairs. The DAG detects write/read dependencies with a register pair granularity:

```
I0=8;
DM(I0,M1)=R1;
```

Certain other sequences of instructions cause incorrect results on the DSP and are flagged as errors by DSP assembler software. These types of instructions can execute on the processor, but cause incorrect results:

- An instruction that stores a DAG register in memory using indirect addressing from the same DAG, with or without update of the index register. The instruction writes the wrong data to memory or updates the wrong index register.
    - **Do not try these:** `DM(M2,I1)=I0;` or `DM(I1,M2)=I0;` These example instructions do not work because I0 and I1 are both DAG1 registers.
- An instruction that loads a DAG register from memory using indirect addressing from the same DAG, with update of the index register. The instruction will either load the DAG register or update the index register, but not both.
    - **Do not try this:** `L2=DM(I1,M0);` This example instruction does not work because both L2 and I1 are DAG1 registers.

DAG Instruction Summary

Table 4-2, Table 4-3, Table 4-4, Table 4-5, Table 4-6, Table 4-7, Table 4-8, and Table 4-9 list the DAG instructions. For more information on assembly language syntax, see the *ADSP-21160 SHARC DSP Instruction Set Reference*. In these tables, note the meaning of the following symbols:

- **I15-8** indicates a DAG2 index register: `I15`, `I14`, `I13`, `I12`, `I11`, `I10`, `I9`, or `I8`, and **I7-0** indicates a DAG1 index register `I7`, `I6`, `I5`, `I4`, `I3`, `I2`, `I1`, or `I0`.
- **M15-8** indicates a DAG2 modify register: `M15`, `M14`, `M13`, `M12`, `M11`, `M10`, `M9`, or `M8`, and **M7-0** indicates a DAG1 modify register `M7`, `M6`, `M5`, `M4`, `M3`, `M2`, `M1`, or `M0`.

- **Ureg** indicates any universal register; For a list of the DSP's universal registers, see Table A-1 on page A-2.
- **Dreg** indicates any data register; For a list of the DSP's data registers, see the Data Register File registers that are listed in Table A-1 on page A-2.
- **Data32** indicates any 32-bit value, and **Data6** indicates any 6-bit value

Table 4-2. Post-Modify Addressing, Modified by M Register and Updating I Register

DM(I7-0,M7-0)=Ureg (LW); {DAG1}
PM(I15-8,M15-8)=Ureg (LW); {DAG2}
Ureg=DM(I7-0,M7-0) (LW); {DAG1}
Ureg=PM(I15-8,M15-8) (LW); {DAG2}
DM(I7-0,M7-0)=Data32; {DAG1}
PM(I15-8,M15-8)=Data32; {DAG2}

Table 4-3. Post-Modify Addressing, Modified by 6-Bit Data and Updating I Register

DM(I7-0,Data6)=Dreg; {DAG1}
PM(I15-8,Data6)=Dreg; {DAG2}
Dreg=DM(I7-0,Data6); {DAG1}
Dreg=PM(I15-8,Data6); {DAG2}

Table 4-4. Pre-Modify Addressing, Modified by M Register (No I Register Update)

DM(M7-0,I7-0)=Ureg (LW); {DAG1}
PM(M15-8,I15-8)=Ureg (LW); {DAG2}
Ureg=DM(M7-0,I7-0) (LW); {DAG1}
Ureg=PM(M15-8,I15-8) (LW); {DAG2}

Table 4-5. Pre-Modify Addressing, Modified by 6-Bit Data (No I Register Update)

DM(Data6,I7-0)=Dreg; {DAG1}
PM(Data6,I15-8)=Dreg; {DAG2}
Dreg=DM(Data6,I7-0); {DAG1}
Dreg=PM(Data6,I15-8); {DAG2}

Table 4-6. Pre-Modify Addressing, Modified by 32-Bit Data (No I Register Update)

Ureg=DM(Data32,I7-0) (LW); {DAG1}
Ureg=PM(Data32,I15-8) (LW); {DAG2}
DM(Data32,I7-0)=Ureg (LW); {DAG1}
PM(Data32,I15-8)=Ureg (LW); {DAG2}

Table 4-7. Update (Modify) I Register, Modified by M Register

Modify(I7-0,M7-0); {DAG1}
Modify(I15-8,M15-8); {DAG2}

Table 4-8. Update (Modify) I Register, Modified by 32-Bit Data

Modify(I7-0,Data32); {DAG1}
Modify(I15-8,Data32); {DAG2}

Table 4-9. Bit-Reverse and Update I Register, Modified by 32-Bit Data

Bitrev(I7-0,Data32); {DAG1}
Bitrev(I15-8,Data32); {DAG2}

5 MEMORY

The DSP contains a large, dual-ported internal memory and provides access to external memory through the DSP's external port. This chapter describes the DSP's memory and how to use it. For information on connecting and timing accesses to external memory, see "External Memory Interface" on page 7-3.

Overview

There are 8 Mbits of internal memory space on the DSP. Within this space, the ADSP-21160 DSP has 4 Mbits of memory, which is divided into two 2 Mbit blocks: Block 0 and Block 1. The remaining, unpopulated 4 Mbits of the memory space are reserved on the ADSP-21160 DSP. Table 5-1 shows the maximum number of data or instruction words that can fit in a 2 Mbit internal memory block.

Table 5-1. Words Per 2 MBit Internal Memory Block

Word Type	Bits Per Word	Maximum Number of Words Per 2 MBit block
Instruction	48-bits	42.67K words
Long Word Data	64-bits	32K words
Extended Precision Normal Word Data	40-bits	42.67K words
Normal Word Data	32-bits	64K words
Short Word Data	16-bits	128K words

There are 4 Gwords of external memory space that the DSP can address. External memory connects to the DSP's external port, which extends the DSP's 32-bit address and 64-bit data buses off the DSP. The DSP can make 64-bit or 32-bit accesses to external memory for instructions or data. Table 5-2 shows the access types and words for DSP external memory accesses. The DSP's DMA controller automatically packs external data into the appropriate word width during data transfer.

Table 5-2. Internal-to-External Memory Word Transfers[1]

Word Type	Transfer Type
Instruction	48-bit word MSB justified within 64-bit transfer
Long Word Data	64-bit word in 64-bit transfer
Extended Precision Normal Word Data	40-bit word MSB justified within 64-bit transfer
Normal Word Data	32-bit word in 32-bit transfer
Short Word Data	Not supported

1 For external port word alignment, see Figure 7-1 on page 7-2

Most microprocessors use a single address and data bus for memory access. This type of memory architecture is called Von Neumann architecture. But, DSPs require greater data throughput than Von Neumann architecture provides, so many DSPs use memory architectures that have separate buses for program and data storage. The two buses let the DSP get a data word and an instruction simultaneously. This type of memory architecture is called Harvard architecture.

SHARC DSPs go a step farther by using a Super Harvard architecture. This architecture has program and data buses, but provides a single, unified address space for program and data storage. While the Data Memory (DM) bus only carries data, the Program Memory (PM) bus handles instructions or data, allowing dual-data accesses.

DSP core and I/O processor accesses to internal memory are completely independent and transparent to one another. Each block of memory can

be accessed by the DSP core and I/O processor in every cycle—no extra cycles are incurred if the DSP core and the I/O processor access the same block.

A memory access conflict can occur when the processor core attempts two accesses to the same internal memory block in the same cycle. When this conflict happens, an extra cycle is incurred. The DM bus access completes first and the PM bus access completes in the following (extra) cycle.

During a single-cycle, dual-data access, the processor core uses the independent PM and DM buses to simultaneously access data from both memory blocks. Though dual-data accesses provide greater data throughput, it is important to note some limitations on how programs may use them. The limitations on single-cycle, dual-data accesses are:

- The two pieces of data must come from different memory blocks.

  If the core tries to access two words from the same memory block (over the same bus) for a single instruction, an extra cycle is needed. For more information on how the buses access these blocks, see "Internal Memory" on page 5-14

- The data access execution may not conflict with an instruction fetch operation.

  If the cache contains the conflicting instruction, the data access completes in a single-cycle and the sequencer uses the cached instruction. If the conflicting instruction is not in the cache, an extra cycle is needed to complete the data access and cache the conflicting instruction. For more information, see "Instruction Cache" on page 3-10.

Efficient memory usage relies on how the program and data are arranged in memory and varies how the program accesses the data. For more information, see "Arranging Data in Memory" on page 5-84.

As shown in Table 5-1, the DSP has three internal buses connected to its dual-ported memory, the Program Memory (PM) bus, Data Memory (DM) bus, and I/O Processor (IO) bus. The PM bus and DM bus share one memory port and the IO bus connects to the other port. Memory accesses from the DSP's core (computational units, data address generators, or program sequencer) use the PM or DM buses, while the I/O processor uses the IO bus for memory accesses. Using the IO bus, the I/O processor can provide data transfers between internal memory and the DSP's communication ports (link ports, serial ports, and external port) without hindering the DSP core's access to memory.

While the DSP's internal memory is divided into **blocks** that can be accessed by DAG1 and DAG2, the DSP's external memory spaces is divided into **banks**, which may be addressed by either data address generator. External memory banks also may be configured for size and access waitstates. "External Memory" on page 5-20

The DSP core's PM bus and DM bus and I/O processor's External Port (EP) bus can try to access multiprocessor memory space or external memory space in the same cycle. The DSP has a two level arbitration system to handle this conflicting access. Arbitration stems from a priority convention and the state of the `SYSCON` register's `EBPRx` bits. When arbitrating between the processor core buses, the DM bus always has priority over the PM bus. Arbitration between the winning core bus and I/O processor EP bus depends on the priority set with the `EBPRx` bits. For more information on setting this priority, see "External Bus Priority" on page 5-33.

Internal Address and Data Buses

Figure 5-1 on page 5-5 also shows that the PM buses, DM buses, and I/O processor have access to the external bus (pins `DATA63-0`, `ADDR31-0`) through the DSP's external port. The external port provides access to system (off-DSP) memory and peripherals. This port also lets the DSP access the internal memory of other DSPs if connected in a multiprocessing system.

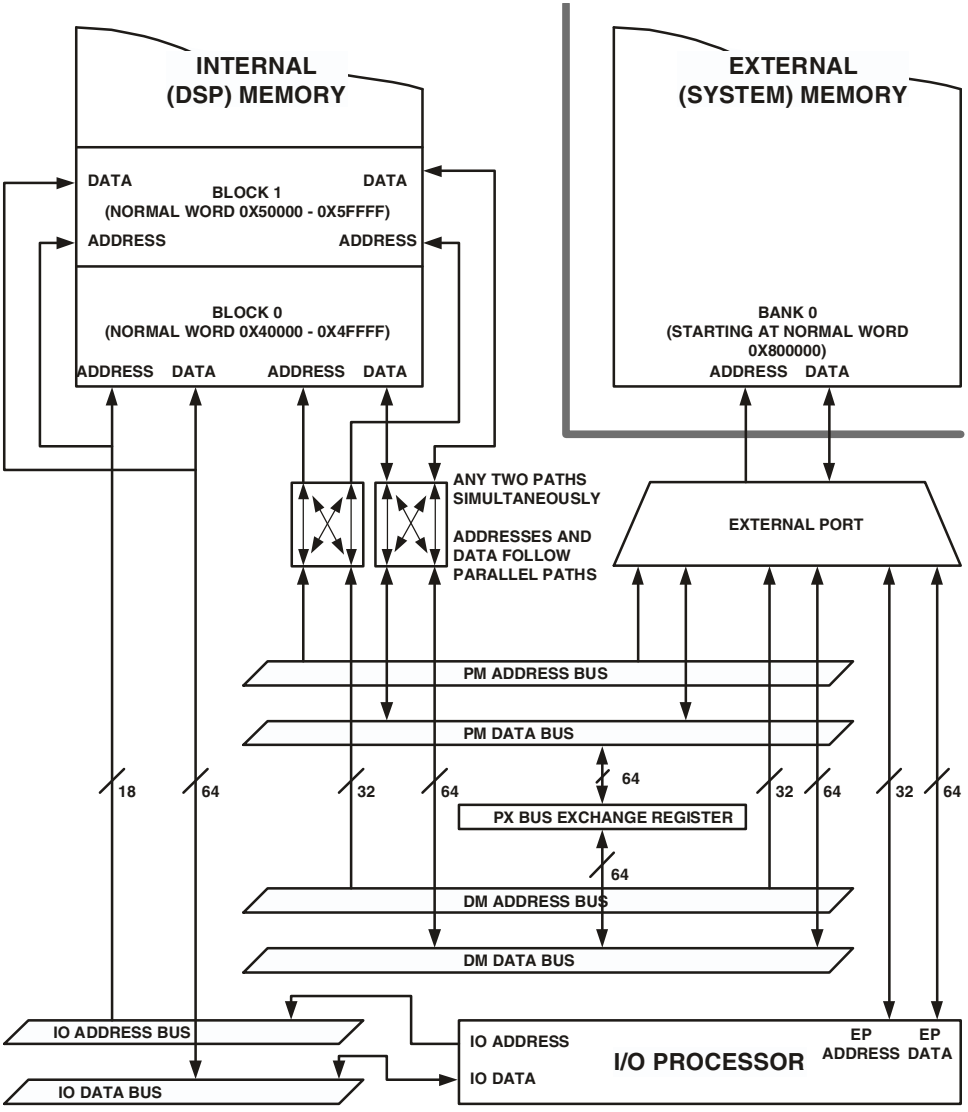


Figure 5-1. ADSP-21160 Memory and Internal Buses Block Diagram

Almost without exception, the DSP's three buses can access all memory spaces, supporting all data sizes. There are three restrictions on the access of buses to memory. The limitations on the PM, DM, and IO buses are as follows:

- The PM, DM, and IO buses may only make Normal Word addressing accesses to multiprocessor or external memory. For more information, see "Multiprocessor Memory" on page 5-17.
- The IO bus may not access the I/O processor's memory mapped registers. For more information, see "I/O Processor"
- The IO bus may not use Short word addressing for DMA operation.

Addresses for the PM and DM buses come from the DSP's program sequencer and Data Address Generators (DAGs). The program sequencer and DAGs supply 32-bit addresses for locations throughout the DSP's memory spaces. The DAGs supply addresses for data reads and writes on both the PM and DM address buses, while the program sequencer uses only the PM address bus for sequencing execution.

Each DAG is associated with a particular data bus. DAG1 supplies addresses over the DM bus and DAG2 supplies addresses over the PM bus. For more information on address generation, see "Program Sequencer" or "Data Address Generators".

Because the DSP's internal memory is arranged in four 16-bit wide by 32K high columns, memory is addressable in widths that are multiples of columns up to 64 bits: 1 column = 16-bit words, 2 columns = 32-bit words, 3 columns = 48- or 40-bit words, and 4 columns = 64-bit words. For more information on the how the DSP works with memory words, see "Memory Organization and Word Size" on page 5-22.

The PM and DM data buses are 64 bits wide. Both data buses can handle Long word (64-bit), Normal word (32-bit), Extended-precision Normal

word (40-bit), and Short word (16-bit) data, but only the PM data bus carries Instruction words (48-bit).

At the processor's external port, the DSP multiplexes the three memory buses—PM, DM, and I/O—to create a single off-chip data bus (DATA 63-0) and address bus (ADDR 31-0).

Internal Data Bus Exchange

The data buses let programs transfer the contents of any register in the DSP to any other register or to any internal memory location in a single cycle. As shown in Figure 5-2, the PM Bus Exchange (PX) register permits data to flow between the PM and DM data buses. The PX register can work as one 64-bit register or as two 32-bit registers (PX1 and PX2). The alignment of PX1 and PX2 within PX appears in Figure 5-3.

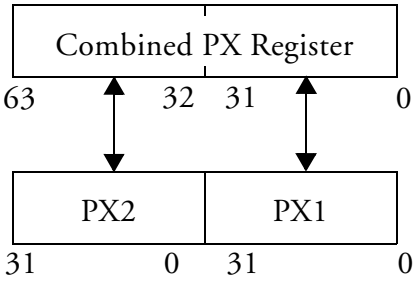


Figure 5-2. PM Bus Exchange (PX, PX1, and PX2) Registers

PX1, PX2, and the combined PX register are Universal registers (UREG) and are accessible for register-to-register or memory-to-register transfers.

PX register-to-register transfers with data registers are either 40-bit transfers for the combined PX or 32-bit transfers for PX1 or PX2. Figure 5-3 shows the bit alignment for these types of transfers.

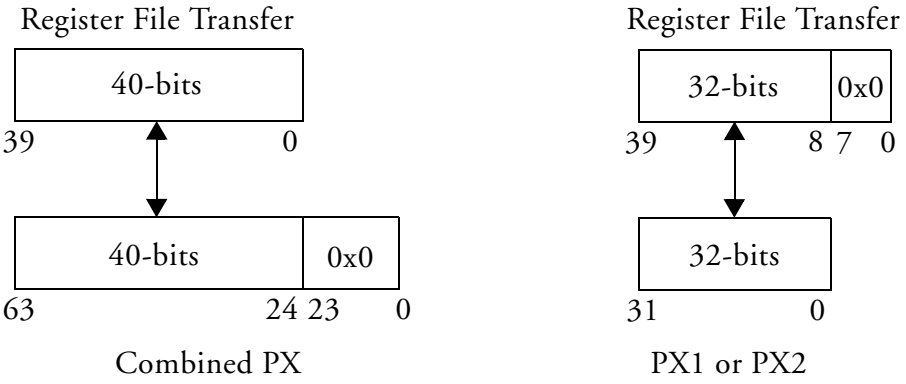


Figure 5-3. PX, PX1, and PX2 Register-to-Register Transfers

Figure 5-3 shows that:

- During a transfer between PX1 or PX2 and a data register file register (DREG), the bus transfers the upper 32 bits of the register file and zero fills the eight LSBs.
- During a transfer between the combined PX register and a register file register, the bus transfers the upper 40 bits of PX and zero fills the lower 24 bits.

PX register-to-memory transfers over the DM data bus are either 64-bit for the combined PX or 32-bit transfers (on bits 31-0 of the bus) for PX1 or PX2. Figure 5-4 shows these transfers.

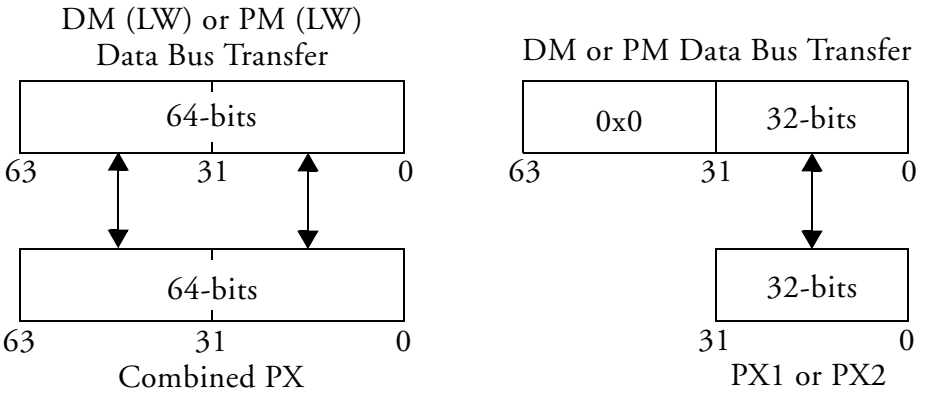


Figure 5-4. PX, PX1, PX2 Register-to-Memory Transfers on DM (LW) or PM (LW) Data Bus

The LW notation in Figure 5-4 draws attention to an important feature of PX register-to-memory transfers over the PM or DM data bus for the combined PX register. PX transfers to memory are 48-bit (3-column) transfers on bits 63-16 of the PM or DM data bus, unless forced to be 64-bit (4-column) transfers with the LW (Long Word) mnemonic.

The status of the memory block's Internal Memory Data Width (IMDWx) setting does not effect this default transfer size for PX to internal memory.

Table 5-5 shows the default transfer size between PX and internal memory over the PM or DM data bus.

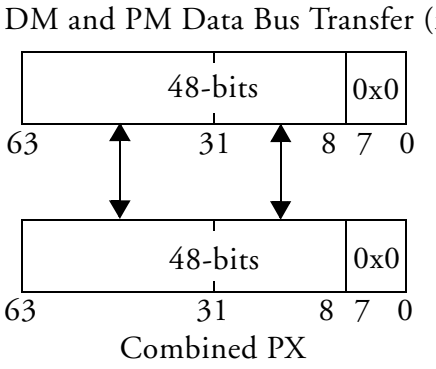


Figure 5-5. PX Register-to-Memory Transfers on PM Data Bus (Without LW)

This default 3-column memory access for the PX register over the PM or DM data bus has a particularly useful application in boot loading. If a program was loading a series of instructions from external memory locations into internal memory, the program could use the following code:

```
i8 = start; {sets up dag2 to auto-increment on PM access}
m8 = 1;
i0 = source; {sets up dag1 to auto-increment on DM access}
m0 = 1;

lcntr = 1000, do external_load until lce; {sets up load loop}
px = dm(i0,m0);
external_load: pm(i8,m8) = px;

/* the loop moves from a range of external addresses */
/* (starting at source) to a range of internal addresses */
/* (starting at start) using the PX register. The external */
/* move is a 64-bit (long word, 4-column) access, and the */
/* internal move is a 48-bit (instruction word, 3-column) */
```

```
/* access. */

jump start; {after load is complete, starts program}
```

Using the PX register for 48-bit moves over the PM data bus can be useful for programs that handle non-standard loading operations. For information on more typical booting techniques, see "I/O Processor". For more information on using DAGs, see "DAG Operations" on page 4-9. For more information on sequencing (Jump and Do/Until), see "Program Sequencer". For more information on how the DSP works with memory words, see "Memory Organization and Word Size" on page 5-22.

For the previous example, a 64-bit transfer to the Port1 memory location can be accomplished by adding the LW bit extension to the data move instruction:

```
PM(Port1)=PX (LW); {move all 64 bits of PX to Port1}
```



The LW bit extension does not alter the data alignment for the PX transfer. Specifying the LW bit extension on a PX write to internal memory generates a write to all four 16-bit columns of the memory destination, rather than three 16-bit columns.

Some transfers over the DM and PM buses have fixed sizes (64- or 48-bit) regardless whether the instruction includes the LW (Long Word) mnemonic. PM and DM data bus transfers that have fixed sizes include:

- All transfers between the PX and external memory are 64-bits transfers.
- All transfers between the PX register and the I/O processor EPBx registers are 64-bit transfers.
- All transfers between the PX register and the I/O processor LBUFx registers are 48-bit transfers (most significant 48-bits of PX).

- All transfers between the PX register (or any other internal register/memory) and any I/O processor register (other than the EPBx or LBUFx) are 32-bit transfers (least significant 32-bits of PX).
- All transfers between the PX register and data registers (R0-R15 or S0-S15) are 40-bit transfers.

There is no implicit move when the combined PX register is used in SIMD mode. For example in SIMD mode, the following moves could occur:

```
PX1 = R0; {R0 32-bit explicit move to PX1,
           and R1 32-bit implicit move to PX2}

PX = R0; {R0 40-bit explicit move to PX,
          but no implicit move for R1}
```

ADSP-21160 DSP Memory Map

The ADSP-21160 DSP's memory map appears in Figure 5-6 and has three memory spaces: internal memory space, multiprocessor memory space, and external memory space.

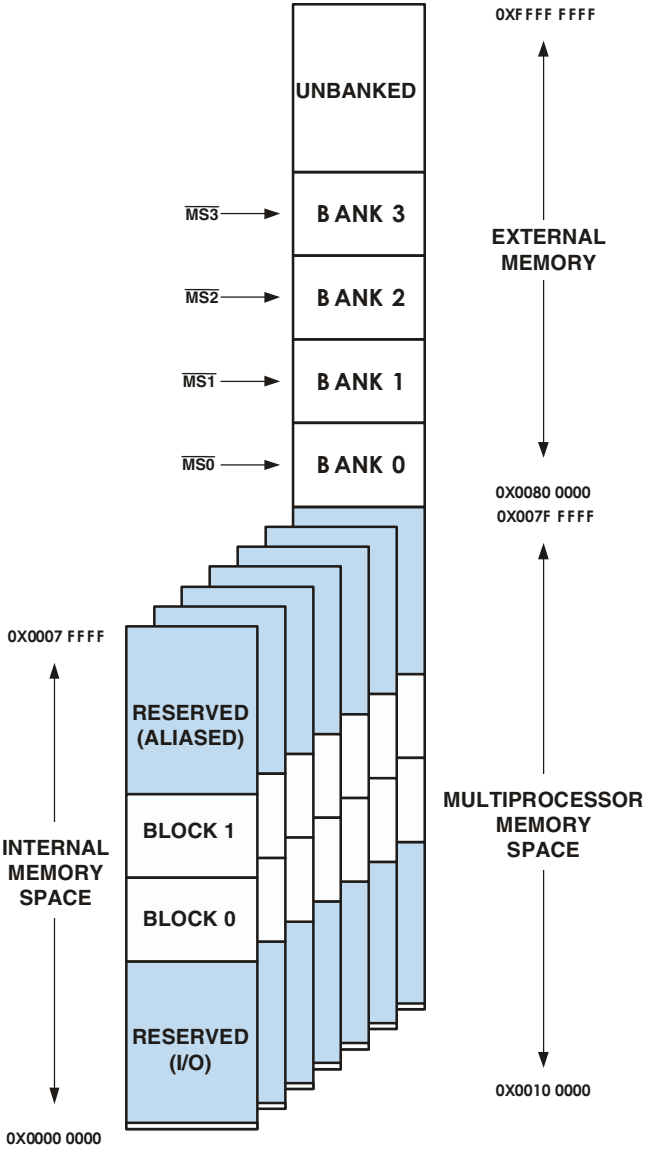


Figure 5-6. ADSP-21160 Memory Map

These spaces have the following definitions:

- **Internal memory space.** This space ranges from address 0x0000 0000 through 0x0007 FFFF (Normal word). Internal memory space refers to the DSP's on-chip SRAM and memory mapped registers.
- **Multiprocessor memory space.** This space ranges from address 0x0010 0000 through 0x007F FFFF (Normal word). Multiprocessor memory space refers to the internal memory space of a group of DSPs that are connected in a multiprocessor system.
- **External memory space.** This space ranges from address 0x0080 0000 through 0xFFFF FFFF (Normal word). External memory space refers to the off-chip memory or memory mapped peripherals that are attached to the DSP's external address (ADDR31-0) and data (DATA63-0) buses.

The address ranges of the three memory spaces correspond to fields within the 32-bit address on the PM, DM, or external address buses. For definitions of these bit fields, see "Program Counter Register (PC)" on page A-28.

Internal Memory

The ADSP-21160 DSP's internal memory space appears in Figure 5-7.

This memory space has four address regions.

- **I/O processor memory mapped registers.** This region ranges from address 0x0000 0000 through 0x0000 00FF (Long word)
- **Reserved (I/O) memory.** This region ranges from address 0x0000 0100 through 0x0001 FFFF (Long word). These addresses are not accessible.

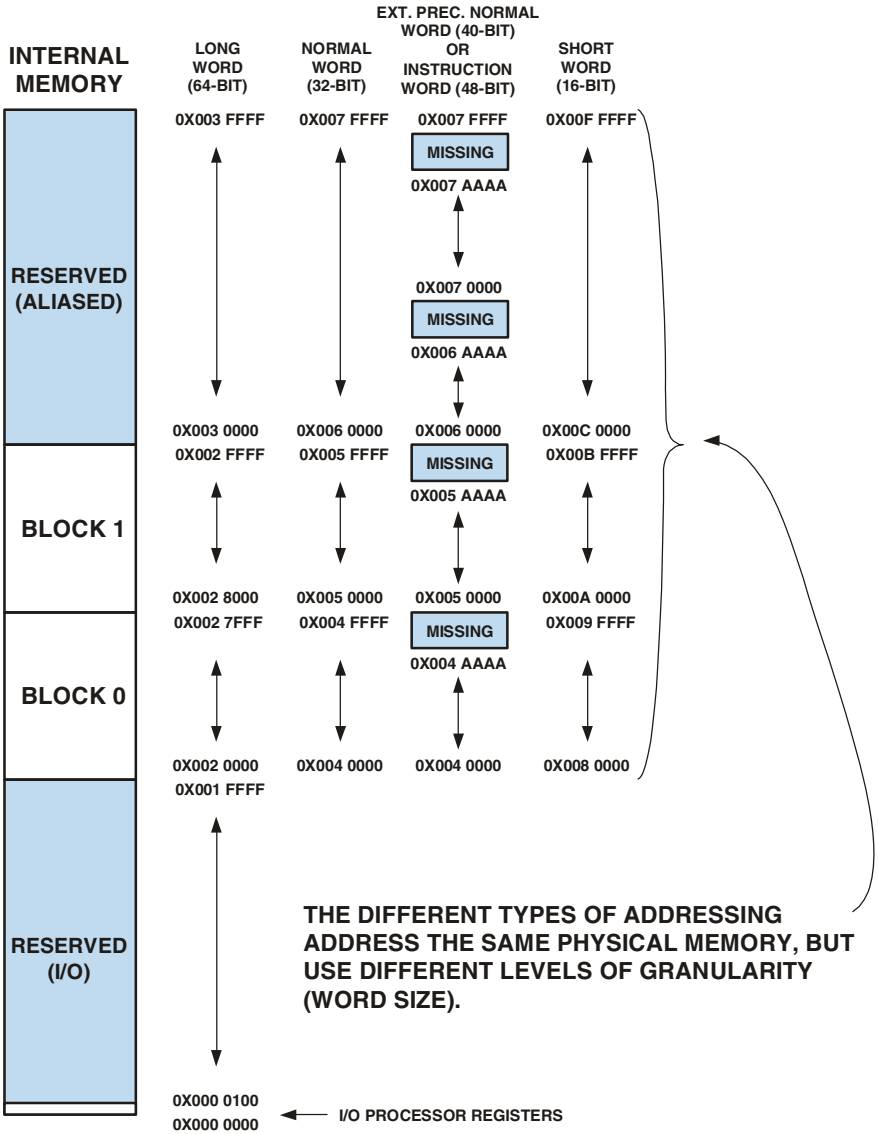


Figure 5-7. ADSP-21160 Internal Memory Space

- **Block 0 memory.** This region ranges from address 0x0002 0000 through 0x0002 7FFF (Long word).
- **Block 1 memory.** This region ranges from address 0x0002 8000 through 0x0002 FFFF (Long word).
- **Reserved (aliased) memory.** This region consists of a Block 0 aliased region from 0x0003 0000 through 0x0003 7FFF and a Block 1 aliased region from 0x0003 08000 through 0x0003 FFF (Long word). Accesses to these two regions result in accesses to the corresponding addresses in Block 0 and Block 1.

The I/O processor's memory-mapped registers control the system configuration of the DSP and I/O operations. For more information, see "I/O Processor" These registers occupy consecutive 32-bit locations in this region.

If a program uses Long word addressing (forced with the LW mnemonic) to accesses this region, the access is only to the addressed 32-bit register, rather than accessing two adjacent I/O processor registers. The register contents are transferred on bits 31-0 of the data bus. There are a couple of exceptions to this one-at-a-time I/O processor register access rule:

- Long word accesses to the external port data buffer locations (EPBx) in SIMD mode access two adjacent 32-bit I/O registers.
- Long word accesses to external port buffer (EPBx) or link port buffer (LBUFx) locations using the PX register access two adjacent 32-bit I/O registers.

As shown in Figure 5-7, the DSP can address memory in the Block 0, Block 1, or the Block 0 and 1 aliased regions using Long word, Normal word, or Short word addressing. The DSP interprets the addressing mode from the address range for the access. Though there are multiple addressing modes for each memory region, these different modes are addressing the same physical memory. For example, the Long word address 0x0002 0000 corresponds to the same locations as Normal word addresses

0x0004 0000 and 0x0004 0001 and corresponds to the same locations as Short word addresses 0x0008 0000, 0x0008 0001, 0x0008 0002, and 0x0008 0003.

Figure 5-7 also shows that there are gaps in the DSP's memory map when using Normal word addressing for 48-bit (instruction word) or 40-bit (extended precision Normal word) accesses. These gaps of missing addresses stem from the arrangement of this 3-column data in memory. For more information, see "Memory Organization and Word Size" on page 5-22.

Multiprocessor Memory

The ADSP-21160's multiprocessor memory space appears in Figure 5-8.

This memory space has seven address regions that correspond to the internal memory of the DSPs in a multiprocessing system. Each of the processors in such a system has a processor ID, which is set with the DSP's ID2-0 pins. The address regions by processor ID are:

- **Internal memory of DSP with ID=001.** This region ranges from address 0x0010 0000 through 0x0017 FFFF.
- **Internal memory of DSP with ID=010.** This region ranges from address 0x0020 0000 through 0x0027 FFFF.
- **Internal memory of DSP with ID=011.** This region ranges from address 0x0030 0000 through 0x0037 FFFF.
- **Internal memory of DSP with ID=100.** This region ranges from address 0x0040 0000 through 0x0047 FFFF.
- **Internal memory of DSP with ID=101.** This region ranges from address 0x0050 0000 through 0x0057 FFFF.

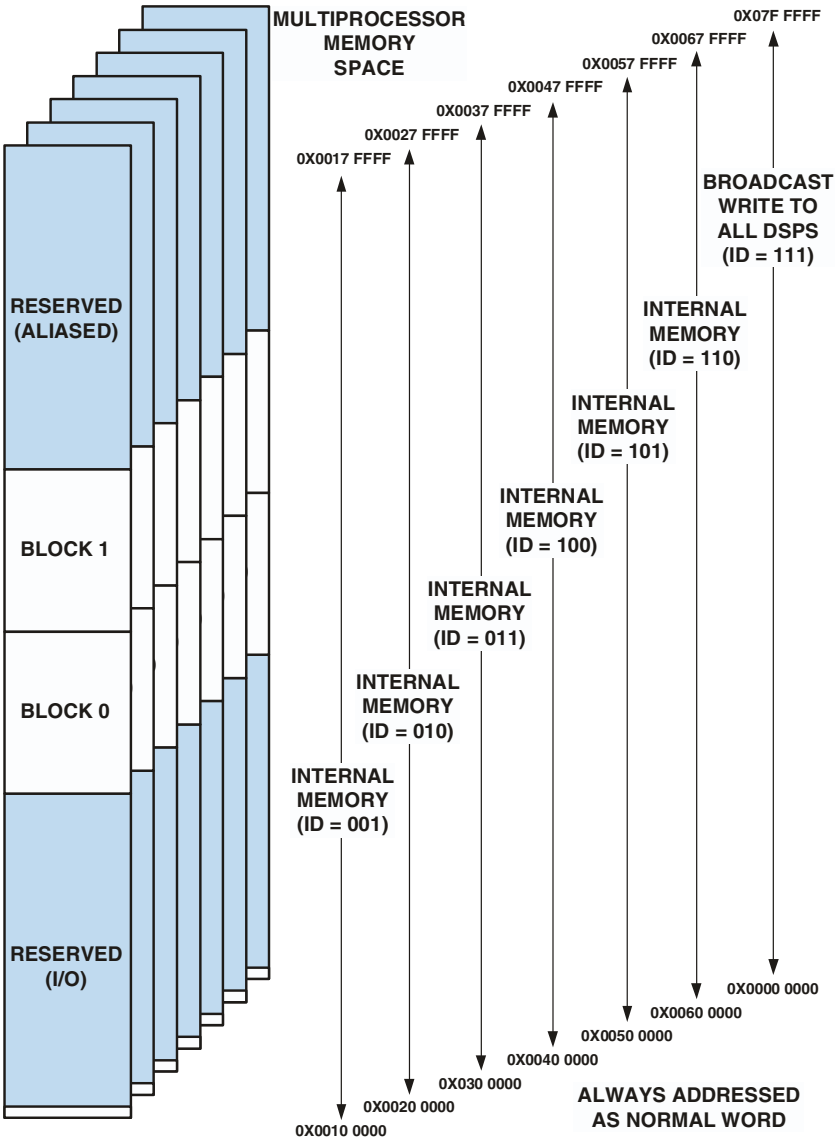


Figure 5-8. ADSP-21160 Multiprocessor Memory Space

- **Internal memory of DSP with ID=110.** This region ranges from address 0x0060 0000 through 0x0067 FFFF.
- **Broadcast write to internal memory of all DSPs (ID=111).** This region ranges from address 0x0070 0000 through 0x0077 FFFF.

It is important to note that programs may only use Normal word addressing in multiprocessor memory space. Long or Short word writes may corrupt valid data, and Long or Short word reads return invalid data.

The address range of the access determines which DSP's internal memory is the multiprocessor memory access source or destination. Broadcast writes (writes in the range 0x0070 0000 through 0x007F FFFF) access the memory of all DSPs in the multiprocessing system.

Instead of using its own internal memory address range, a DSP can access its memory through the DSP's corresponding address range in multiprocessor memory space. In this case, the DSP reads or writes to its own internal memory and does not make an access on the external system bus. Note that such self-accesses through multiprocessor memory space may only be accomplished with processor-core-generated addresses, not I/O processor-generated addresses.

For more information on memory accesses in multiprocessor systems, see "External Port".

External Memory

The ADSP-21160's external memory space appears in Figure 5-9.

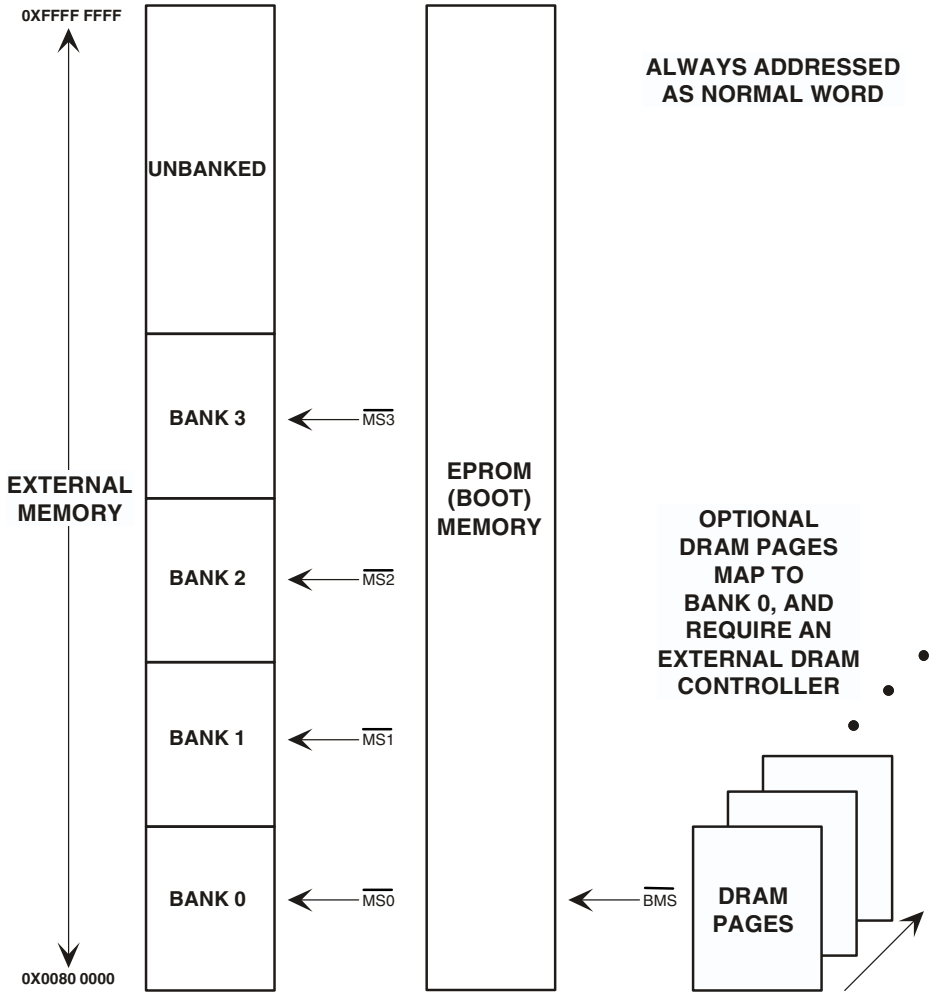


Figure 5-9. ADSP-21160 External Memory Space

The DSP accesses external memory space through the external port, which multiplexes the processor core's PM and DM buses and the I/O processor's EP bus. To address this space, the DSP's DAG1, DAG2, and I/O processor generate 32-bit addresses over the DM, PM, and EP address buses, allowing the DSP access to the complete 4 Gword memory map. But, the program sequencer only generates 24-bit addresses over the PM bus, limiting sequencing to the low 12 Mwords of the memory map.

As shown in Figure 5-9, the external memory space has five regions: 4 banks (Bank 0-3) and an unbanked region. The DSP controls access to the banked regions with memory select lines ($\overline{\text{MS3-0}}$) in addition to the memory address. Access to the unbanked region is controlled only by the memory address. Each region of external memory may be configured for address range and waitstates. For more information on configuring external memory banks, see "Setting Data Access Modes" on page 5-28.

For more information on accessing external memory, see "External Port".

The external memory space can also accommodate an optional boot memory EPROM and optional paged DRAM. For more information, see "Using Boot Memory" on page 5-29 and "External (Bank 0) DRAM Page Size" on page 5-38.

Shadow Write FIFO

Because the DSP's internal memory operates at high speeds, writes to the memory do not go directly into the memory array, but rather to a two-deep FIFO called the shadow write FIFO. This FIFO uses a non-read cycle (either a write cycle, or a cycle in which there is no access of internal memory) to load data from the FIFO into internal memory. When an internal memory write cycle occurs, the FIFO loads any data from a previous write into memory and accepts new data. FIFO operation is normally transparent, but there is one case in which programs need to intervene in the operation of the shadow write FIFO: mixing 48-bit and 32-bit word accesses to the same locations in memory.

The shadow FIFO cannot differentiate between the mapping of 48-bit words and the mapping of 32-bit words. Examples of these mappings appear in Figure 5-10 on page 5-23, Figure 5-11 on page 5-24, Figure 5-12 on page 5-25, and Figure 5-13 on page 5-26. If a program writes a 48-bit word to memory and then tries to read the data with a 16-, 32-, or 64-bit word access or writes a 16-, 32-, 64-bit word to memory and tries to read the data with a 48-bit access, the shadow FIFO does not intercept the read and returns incorrect data.

If a program must mix 48-bit or 40-bit accesses and 16-, 32-, or 64-bit accesses to the same locations, the program must ensure that the FIFO is flushed before attempting to read the data. The program flushes the FIFO by performing two dummy writes or executing two instructions that do not access the internal memory. These operations force the FIFO to automatically use the non-access cycles to push the write data.

Memory Organization and Word Size

The DSP's internal memory is organized as four 16-bit wide by 32K high columns. These columns of memory are addressable as a variety of word sizes:

- 64-bit Long word data (4-columns)
- 48-bit instruction words or 40-bit extended precision Normal word data (3-columns)
- 32-bit Normal word data (2-columns)
- 16-bit Short word data (1-column)



Extended precision Normal word data is left-justified within a 3-column location, using bits 47-8 of the location.

Placing 32-Bit Words and 48-Bit Words

When the processor core or I/O processor addresses memory, the word width of the access determines which columns within the memory are accessed. For instruction words (48-bit) or extended precision Normal word data (40-bit), the word width is 48 bits, and the access selects from the memory's 16-bit columns in groups of three. Because these sets of 3 column accesses are packed in a 4 column matrix, there are four rotations of the columns for storing 48-bit data. The 3-column word rotations within the 4-column matrix appear in Table 5-1 and Figure 5-10.

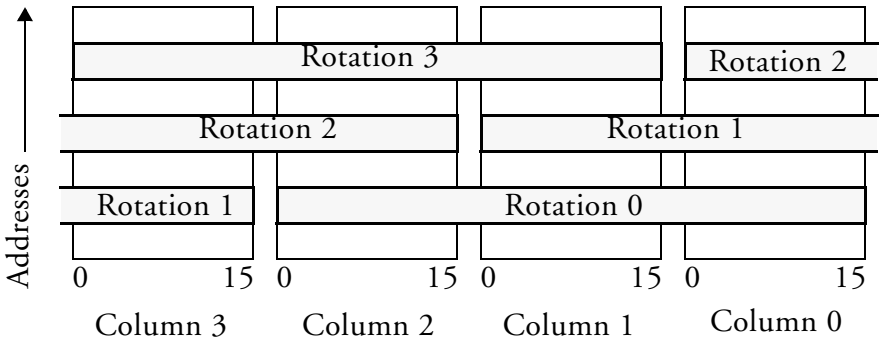


Figure 5-10. 48-bit Word Rotations

For Long word (64-bit), Normal word (32-bit), and Short word (16-bit) memory accesses, The DSP selects from fixed columns in memory. No rotations of words within columns occur for these data types.

Figure 5-7 on page 5-15 shows the memory ranges for each data size in the DSP's internal memory. Figure 5-10 describes 48-bit word rotations.

Mixing 32-Bit and 48-Bit Words

The DSP's memory organization lets programs freely place memory words of all sizes (see "Memory Organization and Word Size" on page 5-22)

with few restrictions (see "Restrictions on Mixing 32-Bit and 48-Bit Words" on page 5-26). This memory organization also lets programs mix (place in adjacent addresses) words of all sizes. This section discusses how to mix odd (3-column) and even (4-column) data words in the DSP's memory.

Transition boundaries between 48-bit (3-column) data and any other data size, can occur at any 64-bit address boundary within either internal memory block. Depending on the ending address of the 48-bit words, there are zero, one, or two empty locations at the transition between the 48-bit (3-column) words and the 64-bit (4-column) words. These empty locations result from the column rotation for storing 48-bit words. The three possible transition arrangements appear in Figure 5-11, Figure 5-12 on page 5-25, and Figure 5-13 on page 5-26.

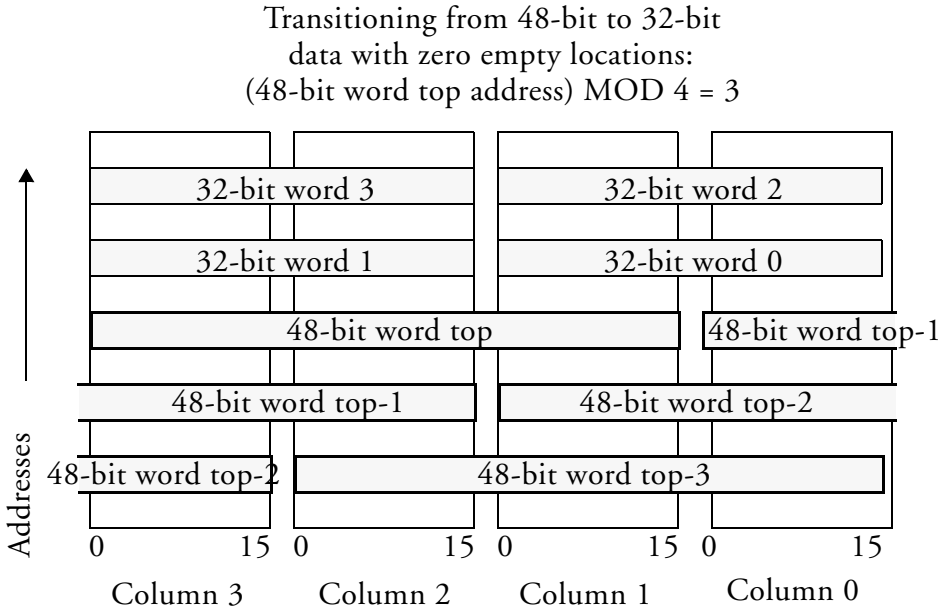


Figure 5-11. Mixed Instructions and Data With No Unused Locations

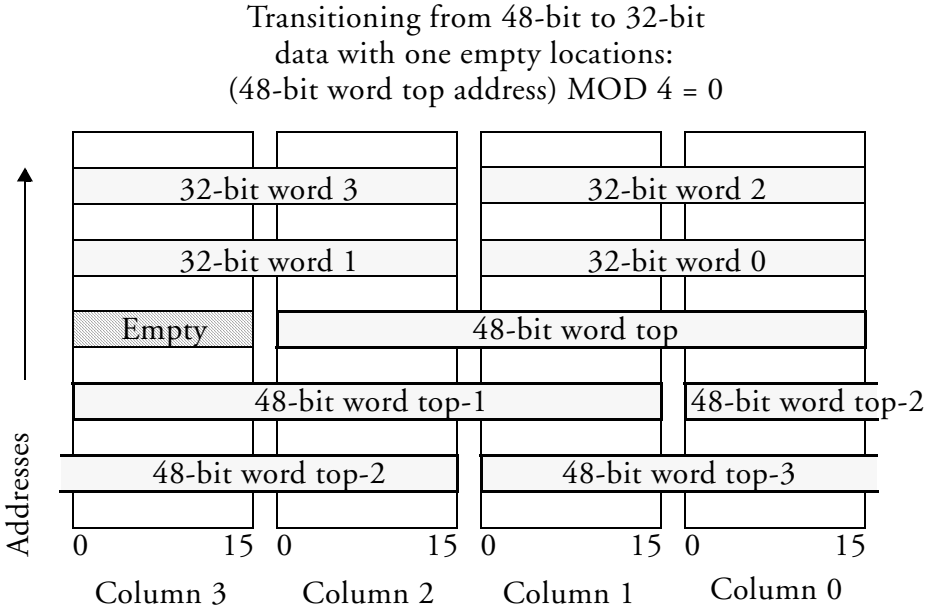


Figure 5-12. Mixed Instructions and Data With One Unused Location

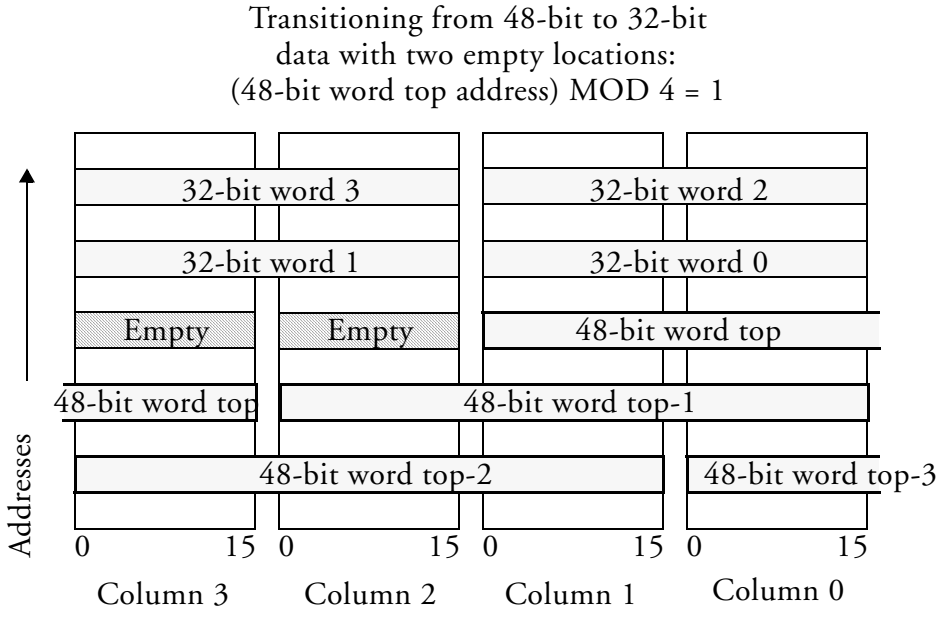


Figure 5-13. Mixed Instructions and Data With Two Unused Locations

Restrictions on Mixing 32-Bit and 48-Bit Words

There are some restrictions that stem from the memory column rotations for 3-column data (48- or 40-bit words) and relate to the way that 3-column data can mix with 4-column data (32-bit words) in memory. These restrictions apply to mixing 48- and 32-bit words, because the DSP uses a Normal word address to access both of these types of data even though 48-bit data maps onto 3-columns of memory and 32-bit data maps onto 2-columns of memory.

When a system has a range of 3-column (48-bit) words followed by a range of 2-column (32-bit) words, there is often a gap of empty 16-bit locations between the two address ranges. The size of the address gap varies with the ending address of the range of 48-bit words. Because the addresses within the gap alias to both 48- and 32-bit words, a 48-bit write

into the gap corrupts 32-bit locations, and a 32-bit write into the gap corrupts 48-bit locations. The locations within the gap are only accessible with Short word (16-bit) accesses.

Calculating the starting address for 4-column data that minimizes the gap after 3-column data is a useful calculation for programs that are mixing 3- and 4-column data.

Given the last address of the 3-column (48-bit) data, the starting address of the 32-bit range that most efficiently uses memory can be determined as follows:

- **n** is the number of contiguous 48-bit words the system has allocated in the internal memory block (**n** < 87,381)
- **B** is the base Normal word address of the internal memory block; if {0 < **n** < 43,691} then **B** = 0x40000 else **B** = 0x50000
- **m** is the first 32-bit Normal word address to use after the end of 48-bit words
    - **m = B + 2 [(n MOD 43,690) – TRUNC((n MOD 43,690) / 4)]**

Another useful calculation for programs that are mixing 3- and 4-column data is to calculate the amount of 3-column data that minimizes the gap before starting 4-column data. Given the starting address of the 4-column (32-bit) data, the number of 48-bit words to allocate that most efficiently uses memory can be determined as follows:

- **m** is the first 32-bit Normal word address after the end of 48-bit words (0x3FFFF < **m** < 0x60000)
- **B** is the base Normal word address of the internal memory block; if {0x3FFFF < **m** < 0x50000} then **B** = 0x40000 else **B** = 0x50000

- **W** is the number of offset words; if {B = 0x50000} then **W** = 43,690 else **W** = 0
- **n** is the number of contiguous 48-bit words the system should allocate in the internal memory block
    - **n = TRUNC{4[(m - B) / 2] / 3]} + W**

Setting Data Access Modes

The SYSCON, MODE1, MODE2, and WAIT registers control the operating mode of the DSP's memory. Table A-17 on page A-45 lists all the bits in SYSCON, Table A-2 on page A-3 lists all the bits in MODE1, Table A-3 on page A-7 lists all the bits in MODE2, and Table A-18 on page A-48 lists all the bits in WAIT. The following bits in SYSCON, MODE1, MODE2, and WAIT registers control memory access modes:

- **Boot Select Override.** SYSCON Bit 1 (BSO) overrides normal usage of $\overline{\text{MSx}}$ chip select lines in favor of the $\overline{\text{BMS}}$ select line for access to boot memory instead of external memory (if 1) or allows normal access to external memory with the $\overline{\text{MSx}}$ chip select lines (if 0).
- **Internal Interrupt Vector Table.** SYSCON Bit 2 (IIVT) forces placement of the interrupt vector table at address 0x0004 0000 regardless of booting mode (if 1) or allows placement of the interrupt vector table as selected by the booting mode (if 0).
- **Internal Memory Block Data Width.** SYSCON Bit 9 (IMDW0) and Bit 10 (IMDW1) select the normal word data access size for internal memory Block 0 and Block1. A block's normal word access size is fixed as 2-column (if IMDWx=0) or 3-column (if IMDWx=1).
- **Memory Bank Size.** SYSCON Bits 15-12 (MSIZE). This bit field selects the size of the four external memory banks (Bank 3-0). The external memory that is not allotted to a bank is part of the Unbanked external memory region.

- **External Bus Priority.** SYSCON Bits 18-17 (EBPRx). This bit field selects the priority for the I/O processor's EP bus when arbitrating access to the DSP's external port.
- **Secondary processor element (PEy).** MODE1 Bit 21 (PEYEN) enables computations in PEy—SIMD mode—(if 1) or disables PEy—SISD mode—(if 0).
- **Broadcast register loads.** MODE1 Bit 22 (BDCST9) and Bit 23 (BDCST1) enable broadcast register loads for memory transfers indexed with I1 (if BDCST1=1) or indexed with I9 (if BDCST9=1).
- **Illegal IOP register access enable.** MODE2 Bit 20 (IIRAE) enables detection of I/O processor register access (if 1) or disables detection (if 0).
- **Unaligned 64-bit memory access enable.** MODE2 Bit 21 (U64MAE) enables detection of uneven address memory access (if 1) or disables detection (if 0).
- **External bank X access mode.** WAIT Bits 1-0 (EB0AM), Bits 6-5 (EB1AM), Bits 11-10 (EB2AM), Bits 16-15 (EB3AM), and Bits 21-20 (UBAM). These bit fields select the access modes for the external memory banks.
- **External bank X waitstates.** WAIT Bits 4-2 (EB0WS), Bits 9-7 (EB1WS), Bits 14-12 (EB2WS), Bits 19-17 (EB3WS), and Bits 24-22 (UBWS). These bit fields select the waitstates for the external memory banks.
- **External bank 0 DRAM page size.** WAIT Bits 27-25 (PAGSZ). This bit field selects the page size for DRAM (allowed in Bank 0 only).

Using Boot Memory

As shown in Figure 5-9 on page 5-20, the DSP supports an external boot EPROM mapped to external memory and selected with the $\overline{\text{BMS}}$ pin. The

boot EPROM provides one of the methods for automatically loading a program in to the internal memory of the DSP after power-up or after a software reset. This process is called booting. For information on boot options and the booting process, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88. For information on systems with a boot EPROM, see "Booting Single and Multiple Processors" on page 11-48.

Reading from Boot Memory

When the DSP boots from an EPROM, the DSP's I/O processor only loads 256 instructions automatically from EPROM. If the whole program must be loaded into internal memory from the EPROM, the DSP must gain access to the boot EPROM after the I/O processor completes the automatic boot process. To manage continuing access to boot memory, the DSP uses the Boot Select Override (BSO) bit in the SYSCON register.

Setting (=1) the BSO bit overrides the external memory selects and asserts the DSP's $\overline{\text{BMS}}$ pin for an external memory DMA transfer. For accessing boot memory, the program first sets the BSO bit in SYSCON then sets up an external port DMA channel to read the EPROM's contents. The program must unmask the DMA channel's interrupt in the IMASK register; if using external port DMA buffer zero (EPOI), the program could enable this interrupt by initializing IMASK to 0x00008003. For more information on external port DMA, see "Setting I/O Processor—EPort Modes" on page 6-13.

While a program may use any external port DMA channel for accessing boot memory, it is important to note that only DMA channel 10 has a special packing mode for boot memory reads. By using DMA channel 10 to complete initial program loading, a program can take advantage of this special packing mode.

When a program sets BSO, the DSP ignores the DMA channel's packing mode (PMODE) bits and forces 8-to-48 bit packing for reads. This special 8-bit packing mode is only available on DMA channel 10 during EPROM

booting or on DMA reads when BSO is set. While one of the external port DMA channels is making a DMA access to boot memory with the BSO bit set, none of the other three channels may make a DMA access to external (not boot) memory.

Only external port DMA transfers assert $\overline{\text{BMS}}$ when BSO is set; processor core accesses to external memory always use the $\overline{\text{MSx}}$ pins. Because the processor core only accesses external (not boot) memory, programs can access external memory in between DMA accesses to boot memory.

Writing to Boot Memory

In systems using write-able EEPROM or FLASH memory for boot memory, programs can write new data to the DSP's boot memory using the boot select override (BSO) pin. As described in "Reading from Boot Memory" on page 5-30, setting (=1) the BSO bit overrides the external memory selects and asserts the DSP's $\overline{\text{BMS}}$ pin for an external memory DMA transfer.

To write to boot memory with the $\overline{\text{BMS}}$ asserted, programs must use DMA channels 11, 12 or 13, but not DMA channel 10. With the BSO bit set, programs should only use DMA channel 10 for reads.

When BSO is set, programs can use DMA channels 11-13 with any settings in channel's the DMACx register, any packing mode, and any data or instruction.



Because boot memory is 8-bits wide and no 8-bit packing mode is available for these writes, programs must use the shifter to place data in the correct location for each write.

Internal Interrupt Vector Table

The default location of the ADSP-21160's interrupt vector table depends on the DSP's booting mode. When the processor boots from an external source (EPROM, host port, or link port booting), the vector table starts at

address 0x0004 0000 (Normal word). When the processor is in "no boot" mode (runs from external memory location 0x0080 0000 without loading), the interrupt vector table starts at address 0x0080 0000.

The Internal Interrupt Vector Table (IIVT) bit in the SYSCON register overrides the default placement of the vector table. If IIVT is set (=1), the interrupt table starts at address 0x0004 0000 (internal memory) regardless of the booting mode.

Internal Memory Block Data Width

The DSP's internal memory blocks use Normal word addressing to access either single-precision 32-bit data or extended-precision 40-bit data. Programs select the data width independently for each internal memory block using the Internal Memory Data Width (IMDW0 and IMDW1) bits in the SYSCON register. If a block's IMDWx bit is cleared (=0), Normal word addressed accesses to the block access Normal word (32-bit) data. If a block's IMDWx bit is set (=1), Normal word addressed accesses to the block access extended precision Normal word (40-bit) data. Reading or writing 40-bit data using a Normal word access to a memory block whose IMDWx bit is cleared (=0) has the following results.

- If a program tries to write 40-bit data (for example, a data register-to-memory transfer), the transfer truncates the lower 8-bits from the register; only writing 32 bits.
- If a program tries to read 40-bit data (for example, a memory-to-data register transfer), the transfer zero-fills the lower 8 bits of the register; only reading 32 bits.

The Program Memory Bus Exchange (PX) register is the only exception to these transfer rules—all loads/stores of the PX register are performed as 48-bit accesses unless forced to 64-bit access with the LW mnemonic. If any 40-bit data must be stored in a memory block configured for 32-bit words, the program should use the PX register to access the 40-bit data in 48-bit words. Programs should take care not to corrupt any 32-bit data

with this type of access. "Restrictions on Mixing 32-Bit and 48-Bit Words" on page 5-26



The Long word (LW) mnemonic only effects Normal word address accesses and overrides all other factors (SIMD, IMDWx).

Memory Bank Size

The DSP's external memory space has four banks of equal, programmable size. The remaining area of external memory that is not assigned to a bank is called unbanked. Mapping peripherals into different banks lets systems accommodate I/O devices with different timing requirements, because the banked and unbanked regions have associated waitstate and access mode settings. For more information, see "External Bank X Access Mode" on page 5-36 and "External Bank X Waitstates" on page 5-37.

As shown in Figure 5-9 on page 5-20, Bank 0 starts at address 0x0080 0000 in external memory, and the Banks 1, 2, 3, and unbanked regions follow. Whenever the DSP generates an address that is located within one of the four banks, the DSP asserts the corresponding memory select line ($\overline{\text{MS3-0}}$).

The size of the memory banks ranges from 8 Kwords to 256 Mwords and is always a power of two. The Memory Size (MSIZE) field of the SYSCON register selects the memory banks size. The value in MSIZE is:

- **MSIZE = log2 (desired bank size in words) - 13**

External Bus Priority

The DSP's internal bus architecture lets the PM bus, DM bus, and I/O processor's EP bus try to access multiprocessor memory space or external memory space in the same cycle. This contending access produces a conflict that the DSP resolves with a two level arbitration policy. The processor core's DM bus always has priority over the PM bus. External Bus Priority (EBPRx) bits in the SYSCON register control the further arbitra-

tion between the winning core bus and the I/O processor's EP bus. The EBPRx field assigns priority as follows:

- If EBPR is 00, priority rotates between core and I/O processor buses.
- If EBPR is 01, the winning core bus has priority over the I/O processor bus.
- If EBPR is 10, the I/O processor bus has priority over the winning core bus.

Secondary Processor Element (PEy)

When the PEYEN bit in the MODE1 register is set (=1), the DSP is in Single-Instruction, Multiple-Data (SIMD) mode. In SIMD mode, many data access operations differ from the DSP's default Single-Instruction, Single-Data (SISD) mode. These differences relate to doubling the amount of data transferred for each data access.

Accesses in SIMD mode transfer both an explicit (named) location and an implicit (un-named, complementary) location. The explicit transfers is a data transfers between the explicit register and the explicit address, and the implicit transfer is between the implicit register and the implicit address.

For information on complementary (implicit) registers in SIMD mode accesses, see "Secondary Processing Element (PEy)" on page 2-36. For more information on complementary (implicit) memory locations in SIMD mode accesses, see "Accessing Memory" on page 5-39.

Broadcast Register Loads

The DSP's BDCST1 and BDCST9 bits in the MODE1 register control broadcast register loading. When broadcast loading is enabled, the DSP writes to complementary registers or complementary register pairs in each processing element on writes that are indexed with DAG1 register I1 (if

BDCST1 =1) or DAG2 register I9 (if BDCST9 =1). Broadcast load accesses are similar to SIMD mode accesses in that the DSP transfers both an explicit (named) location and an implicit (un-named, complementary) location, but broadcast loading only influences writes to registers and write identical data to these registers. Broadcast mode is independent of SIMD mode.

Table 5-3 on page 5-35 shows examples of explicit and implicit effects of broadcast register loads to both processing elements. Note that broadcast loading only effects loads of data registers (register file); broadcast loading does not effect register stores or loads to other system registers. And, broadcast loads only work on register loads; broadcast loading cannot be used for memory writes. For more information on broadcast loading, see "Accessing Memory" on page 5-39.

Table 5-3. Register Load Dual PE Broadcast Operation

Instruction (Explicit, PEx Operation)	**(Implicit, PEy operation)**
Rx = dm(i1,ma); Rx = pm(i9,mb); Rx = dm(i1,ma), Ry = pm(i9,mb);	Sx = dm(i1,ma); Sx = pm(i9,mb); Sx = dm(i1,ma), Sy = pm(i9,mb);

Illegal I/O Processor Register Access

The DSP monitors for I/O processor register access if the Illegal I/O processor Register Access (IIRAE) bit in the MODE2 register is set (=1). When detected, this condition is an input that can cause an Illegal Input Condition Detected (IICDI) interrupt if the interrupt is enabled in the IMASK register.

The I/O processor's DMA controller cannot generate the IICDI interrupt. Only master (not slave) I/O register accesses are detectable. For more information, see "Mode Control 2 Register (MODE2)" on page A-6.

Unaligned 64-bit Memory Access

The DSP monitors for unaligned 64-bit memory accesses if the Unaligned 64-bit Memory Accesses (U64MAE) bit in the MODE2 register is set (=1). An unaligned access is an odd numbered address Normal word access that is forced to 64-bit with the LW mnemonic. When detected, this condition is an input that can cause an Illegal Input Condition Detected (IICDI) interrupt if the interrupt is enabled in the IMASK register. For more information, see "Mode Control 2 Register (MODE2)" on page A-6.

External Bank X Access Mode

The DSP has four modes for accessing external memory space. The External Bank Access Mode (EBxAM) fields in the WAIT register select how the DSP uses waitstates and the acknowledge (ACK) pin to access each external memory bank and unbanked region. The external bank access modes appear in Table 5-4.

Table 5-4. External Bank Access Mode

EBxAM Field	External Bank Access Mode
00	Asynchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change before CLKOUT's edge—accesses use the waitstate count setting from EBxWS and require external acknowledge (ACK), allowing a de-asserted ACK to extend the access time.
01	Synchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change on CLKOUT's edge—reads use the waitstate count setting from EBxWS (minimum EBxWS=001) and require external acknowledge (ACK), allowing a de-asserted ACK to extend the read access time; writes are 0-wait state.
10	Synchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change on CLKOUT's edge—reads use the waitstate count setting from EBxWS (minimum EBxWS=001) and require external acknowledge (ACK), allowing a de-asserted ACK to extend the read access time; writes are 1-wait state.
11	Reserved

External Bank X Waitstates

The DSP applies waitstates to each external memory access depending on the bank's external memory access mode (EBxAM). The External Bank Waitstate (EBxWS) field in the WAIT register sets the number of waitstates for each bank as shown in Table 5-5.

Table 5-5. External Bank Waitstates

EBxWS	# of Waitstates	Hold Time Cycle?[1]
000	0	no
001	1	no
010	2	yes
011	3	yes
100	4	yes
101	5	yes
110	6	yes
111	7	yes

1 Hold Cycle applies to asynchronous mode only.

Table 5-5 lists the hold time settings that EBxWS associates with external memory accesses. A hold time cycle is an inactive bus cycle that the DSP inserts automatically at the end of a read or write, allowing a longer hold time for address and data. The address and data remain unchanged and are driven for one cycle after the DSP deasserts the read or write strobes.



The DSP applies hold time cycles regardless of the external bank access mode (EBxAM). For example, the asynchronous (ACK plus waitstate mode) could also have an associated hold time cycle.

External (Bank 0) DRAM Page Size

As shown in Figure 5-8 on page 5-18, the DSP supports a region of paged DRAM mapped to the Bank 0 region of external memory. Systems placing DRAM in this region require an external DRAM controller to manage page access to the DRAM. For more information, see "DRAM Page Boundary Detection" on page 7-15.

To support DRAM accesses in this region, the DSP detects page boundary crossings and outputs the PAGE signal to the system's DRAM controller. The page boundaries depend on the type of DRAM in the system. For correct operation, programs must configure the page size in the Page Size (PAGESZ) field of the WAIT register. Table 5-6 shows the available PAGESZ settings.

Table 5-6. External DRAM (Bank 0) Page Size

PAGSZ Field	DRAM Page Size
000	256 words
001	512 words
010	1024 words (1K)
011	2048 words (2K)
100	4096 words (4K)
101	8192 words (8K)
110	16384 words (16K)
111	32768 words (32K)

Using Memory Access Status

As described in "Illegal I/O Processor Register Access" on page 5-35 and "Unaligned 64-bit Memory Access" on page 5-36, the DSP can provide illegal access information for Long word or I/O register accesses. When

these conditions occur, the DSP updates an illegal condition flag in a sticky status (STKYx) register. Either of these two conditions can also generate a maskable interrupt. Two ways to use illegal access information are:

- **Interrupts.** Enable interrupts and use an interrupt service routine to handle the illegal access condition immediately. This method is appropriate if it is important to handle all illegal accesses as they occur.
- **STKYx registers.** Use the Bit Tst instruction to examine illegal condition flags in the STKY register after an interrupt to determine which illegal access condition occurred.

Accessing Memory

The word width of DSP processor core accesses to internal memory vary according to the following rules:

- 48-bit access for instruction words, extended precision Normal word (40-bit) data, and PX register
- 64-bit access for Long word data, and Normal word (32-bit) or PX register data with the LW mnemonic
- 32-bit access for Normal word (32-bit) data
- 16-bit access for Short word data

The DSP determines whether a Normal word access is 32- or 40-bit from the internal memory block's IMDWx setting. For more information, see "Internal Memory Block Data Width" on page 5-32. While mixed accesses of 48-bit words and 16-, 32-, or 64-bit words at the same address are not allowed, mixed read/writes of 16-, 32-, and 64-bit words to the same address are allowed. For more information, see "Restrictions on Mixing 32-Bit and 48-Bit Words" on page 5-26.

The DSP's DM and PM buses support 24 combinations of register-to-memory data access options. The following factors influence the data access type:

- Size of words: Short word, Normal word, extended precision Normal word, or Long word
- Number of words: single- or dual-data move
- Mode of DSP: SISD, SIMD, or broadcast load

Access Word Size

The DSP's internal memory accommodates the following word sizes:

- 48-bit instruction words
- 40-bit extended precision Normal word data
- 32-bit Normal word data
- 16-bit Short word data

The DSP's external memory accommodates the following word sizes:

- 48-bit instruction words
- 32-bit Normal word data

To access words of memory, the DSP supports the following memory access word sizes:

- 64-bit accesses, comprised of two consecutive 32-bit data words
- 48-bit accesses, for instruction fetches only
- 40-bit data word accesses

- 32-bit data word accesses
- 16-bit data word accesses

Long Word (64-Bit) Accesses

A program makes a Long word (64-bit) access to internal memory, using an access to a Long word address. Programs can also make a 64-bit access through Normal word addressing with the LW mnemonic or through a PX register move with the LW mnemonic. Programs may not use Long word addressing to access multiprocessor memory space or external memory. The address ranges for internal memory accesses appear in Figure 5-7 on page 5-15.

When data is accessed using Long word addressing, the data is always Long word aligned on 64-bit boundaries in internal memory space. When data is accessed using Normal word addressing and the LW mnemonic, the program should maintain this alignment by using an even Normal word address (least significant bit of address =0). This register selection aligns the Normal word address with a 64-bit boundary (Long word address).

All Long word accesses load or store two consecutive 32-bit data values. The register file source or destination of a Long word access is a set of two neighboring data registers in a processing element. In a forced Long word access (uses the LW mnemonic), the even (Normal word address) location moves to or from the explicit register in the neighbor-pair, and the odd (Normal word address) location moves to or from the implicit register in the neighbor-pair. For example, the following Long word moves could occur:

```
DM(0x40000) = R0 (LW);
{The data in R0 moves to location DM(0x40000),
 and the data in R1 moves to location DM(0x40001).}

R0 = DM(0x40003) (LW);
```

```
{The data at location DM(0x40002) moves to R0,
 and the data at location DM(0x40003) moves to R1.}
```

The example shows that R0 and R1 are a **neighbor** registers in the same processing element. Table 5-7 lists the other neighbor register assignments that apply to Long word accesses.

In un-forced Long word accesses, the DSP places the lower 32-bits of the Long word in the named (explicit) register and places the upper 32-bits of the Long word in the neighbor (implicit) register.

Table 5-7. Neighbor Registers for Long Word Accesses

PEx neighbor registers	PEy neighbor registers
r0 neighbors r1	s0 neighbors s1
r2 neighbors r3	s2 neighbors s3
r4 neighbors r5	s4 neighbors s5
r6 neighbors r7	s6 neighbors s7
r8 neighbors r9	s8 neighbors s9
r10 neighbors r11	s10 neighbors s11
r12 neighbors r13	s12 neighbors s13
r14 neighbors r15	s14 neighbors s15

Programs can monitor for unaligned 64-bit accesses by enabling the U64MAE bit. For more information, see “Unaligned 64-bit Memory Access” on page 5-36.



The Long word (LW) mnemonic only effects Normal word address accesses and overrides all other factors (PEYEN, IMDWx).

Instruction Word (48-Bit) and Extended Precision Normal Word (40-Bit) Accesses

The sequencer uses a 48-bit memory access for instruction fetches. Program can make 48-bit accesses with PX register moves, which default to 48-bit unless the LW mnemonic is part of the instruction.

A program makes an extended precision Normal word (40-bit) access to internal memory using an access to a Normal word address when that internal memory block's IMDWx bit is set (=1) for 40-bit words. Programs may not use extended precision Normal word addressing to access multiprocessor memory space or external memory. The address ranges for internal memory accesses appear in Figure 5-7 on page 5-15. For more information on configuring memory for extended precision Normal word accesses, see "Internal Memory Block Data Width" on page 5-32.

The DSP transfers the 40-bit data to internal memory as a 48-bit value, zero-filling the least significant 8 bits on stores and truncating these 8 bits on loads. The register file source or destination of such an access is a single 40-bit data register.

Normal Word (32-Bit) Accesses

A program makes a Normal word (32-bit) access to internal memory using an access to a Normal word address when that internal memory block's IMDWx bit is cleared (=0) for 32-bit words. Programs use Normal word addressing to access all DSP memory spaces: internal, multiprocessor, and external memory space. The address ranges for memory accesses appear in Figure 5-7 on page 5-15, Figure 5-8 on page 5-18, and Figure 5-9 on page 5-20.

The register file source or destination of a Normal word access is a single 40-bit data register. The DSP zero-fills the least significant 8 bits on loads and truncates these bits on stores.



External memory space accesses using Normal word addressing and the LW mnemonic performs a forced 64-bit access.

Short Word (16-Bit) Accesses

A program makes a Short word (16-bit) access to internal memory, using an access to a Short word address. Programs may not use Short word addressing to access multiprocessor memory space or external memory. The address ranges for internal memory accesses appear in Figure 5-7 on page 5-15.

The register file source or destination of such an access is a single 40-bit data register. The DSP zero-fills the least significant 8 bits on loads and truncates these bits on stores. Depending on the value of the SSE bit in the MODE1 system register, the DSP loads the register's upper 16 bits by either:

- Zero-filling these bits if SSE=0
- Sign-extending these bits if SSE=1

SISD, SIMD, and Broadcast Load Modes

These three processing element modes influence memory accesses. For a comparison of their effects, see the examples in "Data Access Options" on page 5-45. For more information on SISD and SIMD modes, see "Secondary Processing Element (PEy)" on page 2-36.

Broadcast load mode is a hybrid between SISD and SIMD modes, transferring dual-data under special conditions. For examples of broadcast transfers, see "Data Access Options" on page 5-45. For more information on broadcast load mode, see "Broadcast Register Loads" on page 5-34.

Single-and Dual-Data Accesses

The number of transfers that occur in a cycle influences the data access operation. As described on "Overview" on page 5-1, the DSP supports single-cycle, dual-data accesses to internal memory for register-to-memory transfers. Though only available for transfers to data registers, dual-data transfers are extremely useful, because they double the data throughput over single-data transfers. For examples of data flow paths for single- and dual-data transfers, see "Data Access Options" on page 5-45.

Data Access Options

Table 5-8 lists the DSP's 24 possible memory transfer modes that stem from the DSP's data access options. When looking at Table 5-8, it is

important to note that Long and Short word addressing may not target multiprocessor memory space or external memory space.

Table 5-8. The 24 Possible Memory Transfer Modes

Access Type	DSP Mode	Address Space							
		Long Word		Extended Precision		Normal Word		Short Word	
		PM	DM	PM	DM	PM	DM	PM	DM
Single Data Access	SISD mode	LW	none	EW	none	NW	none	SW	none
		none	LW	none	EW	none	NW	none	SW
	SIMD mode	LW	none	EW	none	LW	none	SWx2	none
		none	LW	none	EW	none	LW	none	SWx2
	B-cast Load	LW	none	EW	none	NW	none	SW	none
		none	LW	none	EW	none	NW	none	SW
Dual Data Access	SISD mode	LW	LW	EW	EW	NW	NW	SW	SW
	SIMD mode	LW	LW	EW	EW	LW	LW	SWx2	SWx2
	B-cast Load	LW	LW	EW	EW	NW	NW	SW	SW

Symbols: LW = 64-bit data value (two 32-bit values), EW = 40-bit data value (48-bit value), NW = 32-bit data value, SW = 16-bit data value, and SWx2 = two 16-bit data values.

Table 5-8 shows the transfer modes that stem from the following data access options:

- The mode of the DSP: SISD, SIMD, or Broadcast Load
- The size of access words: Long, extended precision Normal word, Normal word, or Short word
- The number of transferred words: single- or dual-data

Table 5-8 provides a cross reference to examples of each memory access option.

Table 5-9. Memory Transfer Modes Cross Reference

Access Type	DSP Mode	Address Space			
		Long Word	Extended Precision	Normal Word	Short Word
Single Data Access	SISD mode	on page 5-69	on page 5-62	on page 5-55	on page 5-47
	SIMD mode	on page 5-69	on page 5-64	on page 5-57	on page 5-48
	B-cast Load	Figure 5-23	Figure 5-23	Figure 5-21	Figure 5-16
Dual Data Access	SISD mode	on page 5-69	on page 5-59	on page 5-59	on page 5-51
	SIMD mode	on page 5-69	on page 5-64	on page 5-59	on page 5-53
	B-cast Load	Figure 5-26	Figure 5-24	Figure 5-21	Figure 5-17

Short Word Addressing of Single Data in SISD Mode

Figure 5-14 on page 5-49 displays one possible SISD mode, single data, Short word addressed access. For Short word addressing, the DSP treats the data buses as four 16-bit Short word lanes. The 16-bit value for the Short word access transfers using the least significant Short word lane of the PM or DM data bus. The DSP drives the other Short word lanes of the data buses with zeros.

In Figure 5-14 on page 5-49, the access targets PEx registers in a SISD mode operation. This case accesses `WORD X0` whose Short word address has “00” for its least significant two bits of address. Other accesses within this 4-column location have addresses with least significant two bits of “01”, “10”, or “11” and select `WORD X1`, `WORD X2`, or `WORD X3` from memory

respectively. The syntax targets register, RX, in PEx. The example would target a PEy register if using the syntax SX.

The cross (†) in Figure 5-14 indicates that the DSP zero-fills or sign-extends the most significant 16 bits of the data register while loading the Short word value into a 40-bit data register. The selection depends on the state of the SSE bit in the MODE1 system register. The least significant 8 bits of the data register are always zero.

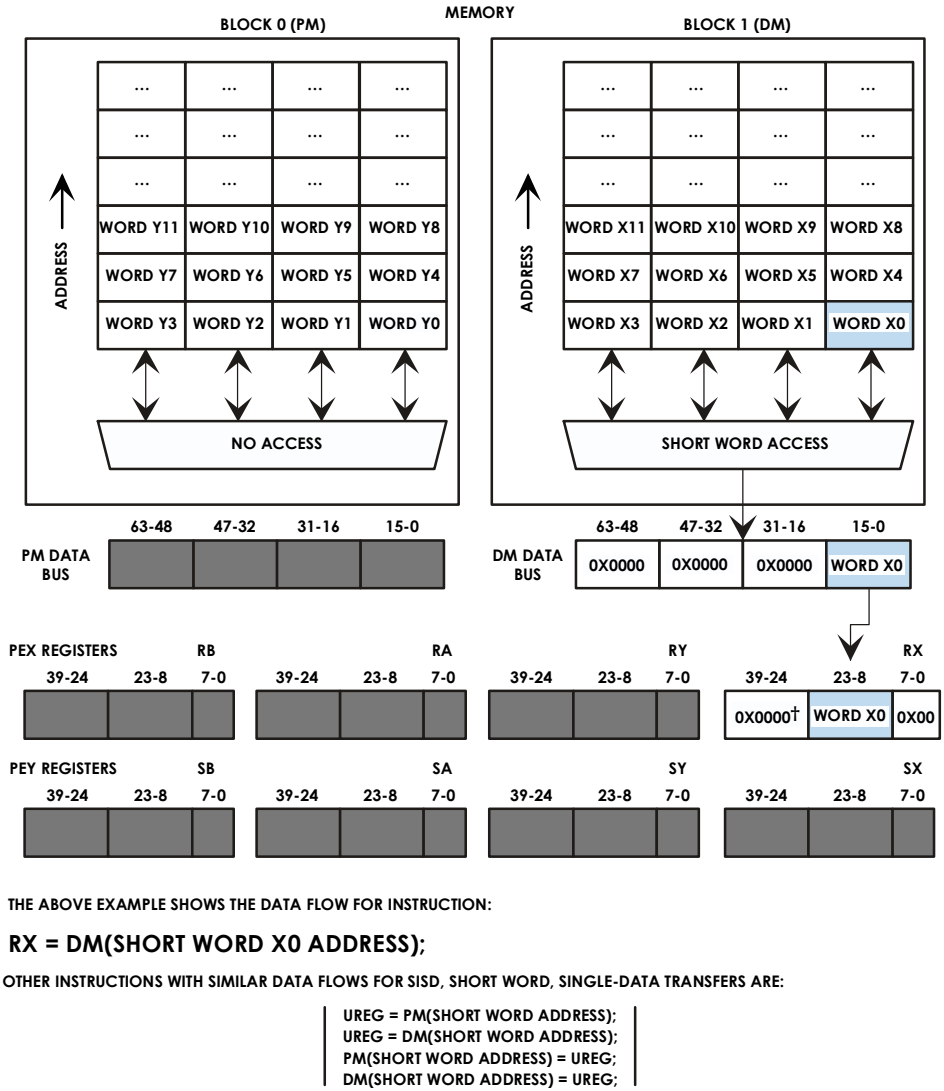
Short Word Addressing of Single Data in SIMD Mode

Figure 5-15 displays one possible SIMD mode, single data, Short word addressed access.

For Short word addressing, the DSP treats the data buses as four 16-bit Short word lanes. The explicitly addressed (named in the instruction) 16-bit value transfers using the least significant Short word lane of the PM or DM data bus. The implicitly addressed (not named in the instruction, but inferred from the address in SIMD mode) Short word value transfers using the 47-32 bit Short word lane of the PM or DM data bus. The DSP drives the other Short word lanes of the PM or DM data buses with zeros.

In Figure 5-15, the explicit access targets the named register, RX, and the implicit access targets that register's complementary register, SX. This case uses a PEx register with an RX mnemonic. If the syntax named a PEy register, SX, as the explicit target the DSP would use that register's complement, RX, as the implicit target. For more information on complementary registers, see "Secondary Processing Element (PEy)" on page 2-36.

The cross (†) in Figure 5-15 indicates that the DSP zero-fills or sign-extends the most significant 16 bits of the data register while loading the Short word value into a 40-bit data register. The selection depends on the state of the SSE bit in the MODE1 system register. The least significant 8 bits of the data register are always zero.



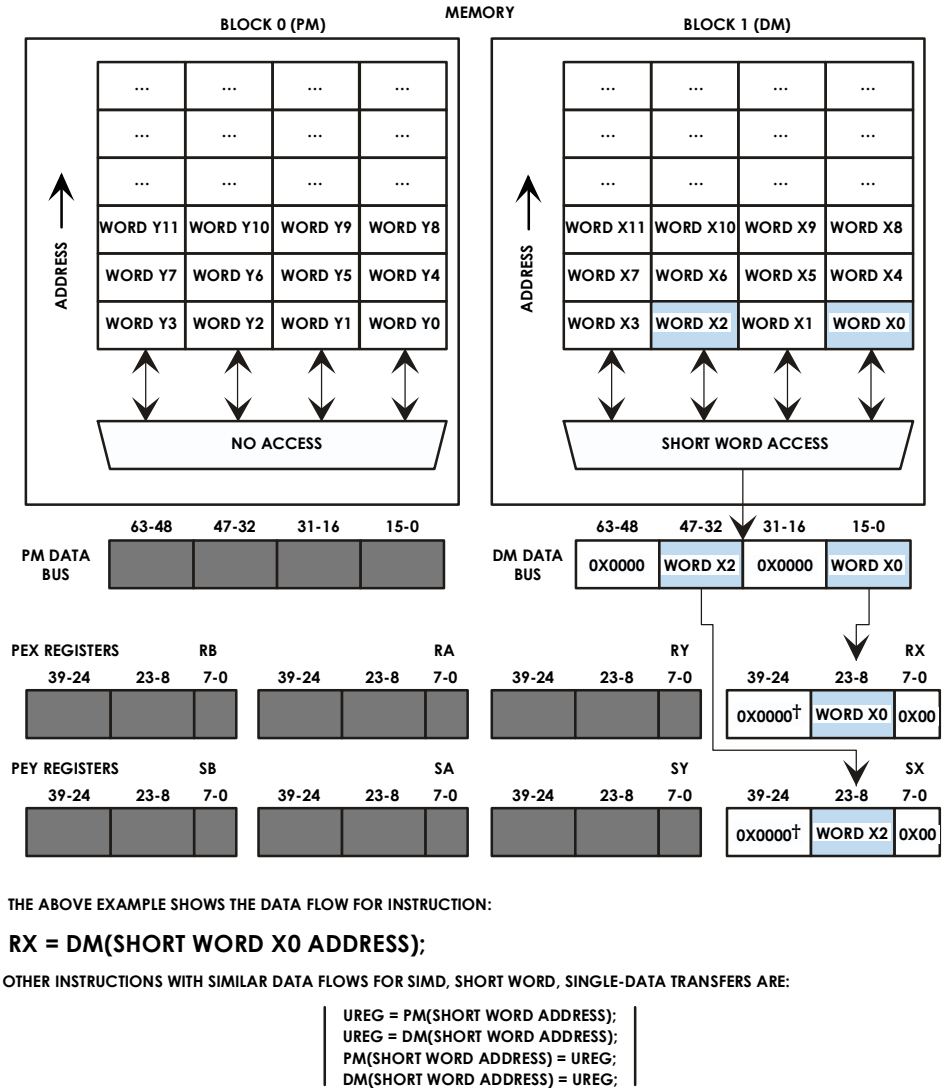
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, SHORT WORD, SINGLE-DATA TRANSFERS ARE:

```
UREG = PM(SHORT WORD ADDRESS);
UREG = DM(SHORT WORD ADDRESS);
PM(SHORT WORD ADDRESS) = UREG;
DM(SHORT WORD ADDRESS) = UREG;
```

Figure 5-14. Short Word Addressing of Single Data in SISD Mode


MEMORY
BLOCK 0 (PM)
BLOCK 1 (DM)
ADDRESS
WORD Y11 WORD Y10 WORD Y9 WORD Y8
WORD Y7 WORD Y6 WORD Y5 WORD Y4
WORD Y3 WORD Y2 WORD Y1 WORD Y0
NO ACCESS
WORD X11 WORD X10 WORD X9 WORD X8
WORD X7 WORD X6 WORD X5 WORD X4
WORD X3 WORD X2 WORD X1 WORD X0
SHORT WORD ACCESS
PM DATA BUS
DM DATA BUS
63-48 47-32 31-16 15-0
0X0000 WORD X2 0X0000 WORD X0
PEX REGISTERS
PEY REGISTERS
RB RA RY RX
SB SA SY SX
39-24 23-8 7-0
0X0000† WORD X0 0X00
0X0000† WORD X2 0X00

THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, SHORT WORD, SINGLE-DATA TRANSFERS ARE:

```
UREG = PM(SHORT WORD ADDRESS);
UREG = DM(SHORT WORD ADDRESS);
PM(SHORT WORD ADDRESS) = UREG;
DM(SHORT WORD ADDRESS) = UREG;
```

Figure 5-15. Short Word Addressing of Single Data in SIMD Mode

Figure 5-15 shows the data path for one transfer, but in this mode it is also important to note the pattern of iterative accesses. For Short word accesses, the DSP accesses Short words sequentially in memory. Table 5-10 shows the pattern of SIMD mode Short word accesses. For more information on arranging data in memory to take advantage of this access pattern, see "Arranging Data in Memory" on page 5-84.

Table 5-10. Short Word Addressing in SIMD Mode

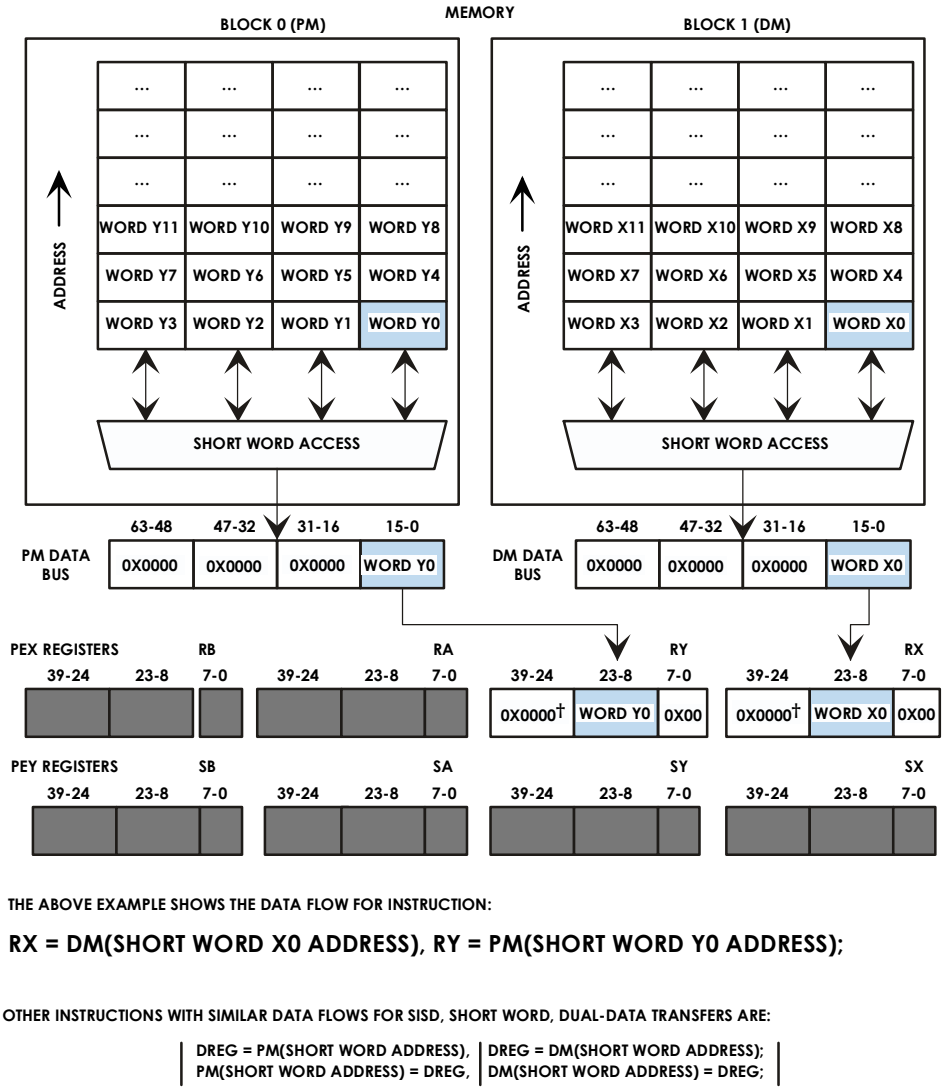
Explicit Short Word Accessed	Implicit Short Word Accessed
Word X0 (Address two LSBs = 00)	Word X2 (Address two LSBs = 10)
Word X1 (Address two LSBs = 01)	Word X3 (Address two LSBs = 11)
Word X2 (Address two LSBs = 10)	Word X4 (Address two LSBs = 00)
Word X3 (Address two LSBs = 11)	Word X5 (Address two LSBs = 01)

Short Word Addressing of Dual-Data in SISD Mode

Figure 5-16 displays one possible SISD mode, dual-data, Short word addressed access.

For Short word addressing, the DSP treats the data buses as four 16-bit Short word lanes. The 16-bit values for Short word accesses transfer using the least significant Short word lanes of the PM and DM data buses. The DSP drives the other Short word lanes of the data buses with zeros. Note that the accesses on both buses do not have to be the same word width. SISD mode dual-data accesses can handle any combination of Short word, Normal word, extended precision Normal word, or Long word accesses. For more information, see "Mixed Word Width Addressing of Dual Data in SISD Mode" on page 5-72.

In Figure 5-16, the access targets PEx registers in a SISD mode operation. This case accesses `WORD XO` in block 1 and `WORD YO` in block 0. Each of these words has a Short word address with "00" for its least significant two bits of address. Other accesses within these 4-column location have the addresses with least significant two bits of "01", "10", or "11" and select


MEMORY
BLOCK 0 (PM)
BLOCK 1 (DM)
ADDRESS
WORD Y11 WORD Y10 WORD Y9 WORD Y8
WORD Y7 WORD Y6 WORD Y5 WORD Y4
WORD Y3 WORD Y2 WORD Y1 WORD Y0
WORD X11 WORD X10 WORD X9 WORD X8
WORD X7 WORD X6 WORD X5 WORD X4
WORD X3 WORD X2 WORD X1 WORD X0
SHORT WORD ACCESS
63-48 47-32 31-16 15-0
PM DATA BUS
DM DATA BUS
0X0000 0X0000 0X0000 WORD Y0
0X0000 0X0000 0X0000 WORD X0
PEX REGISTERS
PEY REGISTERS
RB RA RY RX
SB SA SY SX
39-24 23-8 7-0
0X0000† WORD Y0 0X00
0X0000† WORD X0 0X00

THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS), RY = PM(SHORT WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, SHORT WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(SHORT WORD ADDRESS), PM(SHORT WORD ADDRESS) = DREG,	DREG = DM(SHORT WORD ADDRESS); DM(SHORT WORD ADDRESS) = DREG;

Figure 5-16. Short Word Addressing of Dual Data in SISD Mode

WORD X/Y1, WORD X/Y2, or WORD X/Y3 from memory respectively. The syntax targets registers, RX and RY, in PEx. The example would target PEy registers if using the syntax SX or SY.

The cross (†) in Figure 5-16 indicates that the DSP zero-fills or sign-extends the most significant 16 bits of the data register while loading a Short word value into a 40-bit data register. The selection depends on the state of the SSE bit in the MODE1 system register. The least significant 8 bits of the data register are always zero.

Short Word Addressing of Dual-Data in SIMD Mode

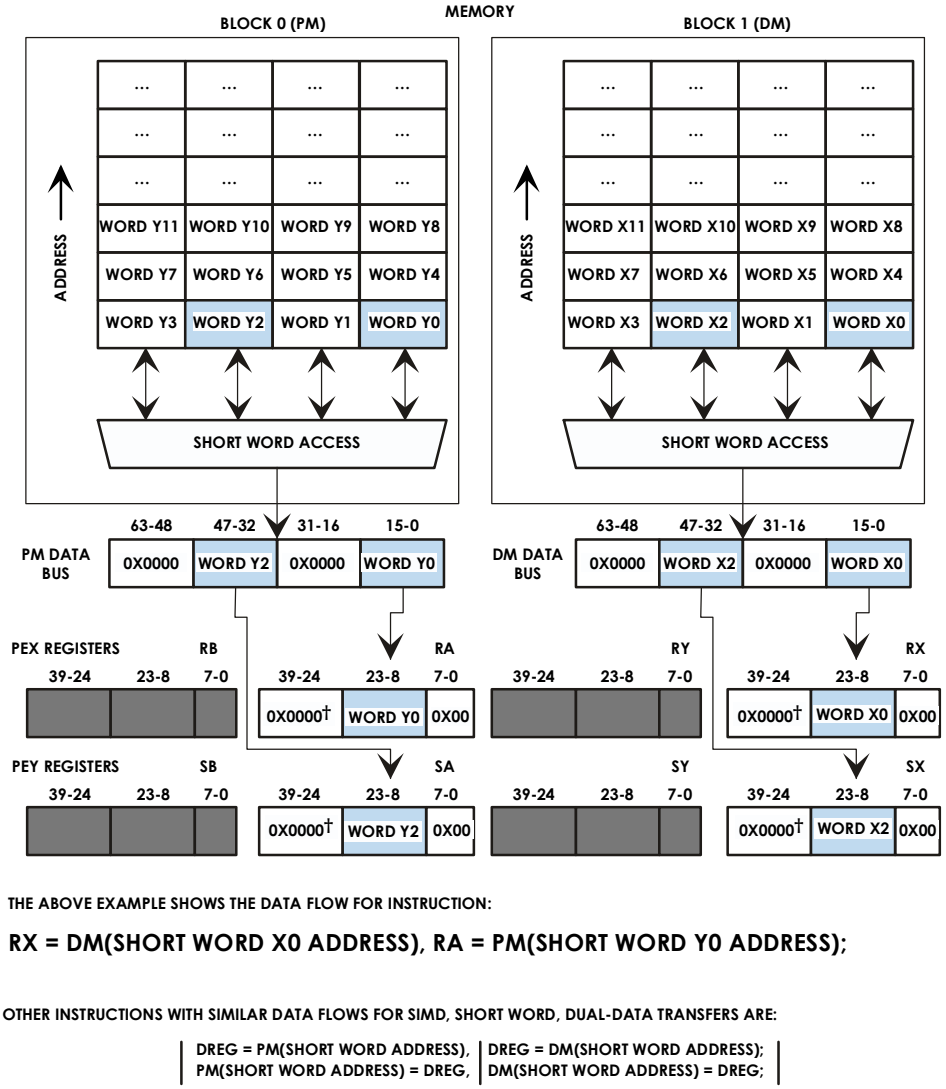
Figure 5-17 displays one possible SIMD mode, dual-data, Short word addressed access.

For Short word addressing, the DSP treats the data buses as four 16-bit Short word lanes. The explicitly addressed (named in the instruction) 16-bit values transfer using the least significant Short word lanes of the PM and DM data bus. The implicitly addressed (not named in the instruction, but inferred from the address in SIMD mode) Short word values transfer using the 47-32 bit Short word lanes of the PM and DM data buses. The DSP drives the other Short word lanes of the PM and DM data buses with zeros.



The accesses on both buses do not have to be the same word width. SIMD mode dual-data accesses can handle combinations of Short word and Normal word or extended precision Normal word and Long word accesses. For more information, see “Mixed Word Width Addressing of Dual Data in SIMD Mode” on page 5-74.

In Figure 5-17, the explicit accesses targets the named registers RX and RA, and the implicit accesses target those register’s complementary registers, SX and SA. This case uses a PEx registers with the RX and RA mnemonics. If the syntax named PEy registers SX and SA as the explicit targets, the DSP would use those registers’ complements, RX and RA, as the implicit targets.



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS), RA = PM(SHORT WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, SHORT WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(SHORT WORD ADDRESS), PM(SHORT WORD ADDRESS) = DREG,	DREG = DM(SHORT WORD ADDRESS); DM(SHORT WORD ADDRESS) = DREG;

Figure 5-17. Short Word Addressing of Dual Data in SIMD Mode

For more information on complementary registers, see “Secondary Processing Element (PEy)” on page 2-36.

The cross (†) in Figure 5-17 indicates that the DSP zero-fills or sign-extends the most significant 16 bits of the data registers while loading the Short word values into the 40-bit data registers. The selection depends on the state of the SSE bit in the MODE1 system register. The least significant 8 bits of the data register are always zero.

Figure 5-17 shows the data path for one transfer, but in this mode it is also important to note the pattern of iterative accesses. For Short word accesses, the DSP accesses Short words sequentially in memory. Table 5-10 on page 5-51 shows the pattern of SIMD mode Short word accesses. For more information on arranging data in memory to take advantage of this access pattern, see “Arranging Data in Memory” on page 5-84.

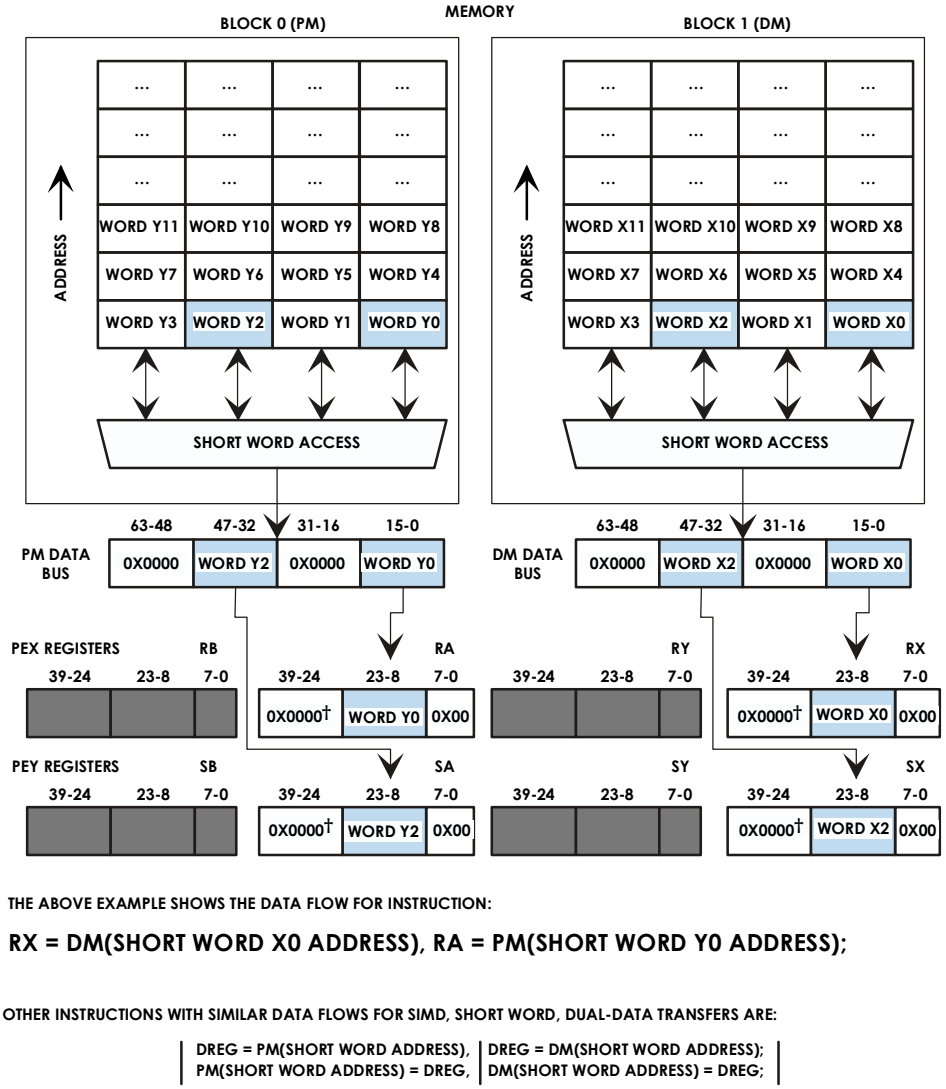
32-Bit Normal Word Addressing of Single Data in SISD Mode

Figure 5-18 displays one possible SISD mode, single data, 32-bit normal word addressed access.

For Normal word addressing, the DSP treats the data buses as two 32-bit Normal word lanes. The 32-bit value for the Normal word access transfers using the least significant Normal word lane of the PM or DM data bus. The DSP drives the other Normal word lanes of the data buses with zeros.

In Figure 5-18, the access targets a PEx register in a SISD mode operation. This case accesses WORD X0 whose Normal word address has “0” for its least significant address bit. The other access within this 4-column location has an addresses with a least significant bit of “1” and selects WORD X1 from memory. The syntax targets register RX in PEx. The example would target a PEy register if using the syntax SX.

For Normal word accesses, the DSP zero-fills least significant 8 bits of the data register on loads and truncates these bits on stores to memory.



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS), RA = PM(SHORT WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, SHORT WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(SHORT WORD ADDRESS), PM(SHORT WORD ADDRESS) = DREG,	DREG = DM(SHORT WORD ADDRESS); DM(SHORT WORD ADDRESS) = DREG;

Figure 5-18. Normal Word Addressing of Single Data in SISD Mode

32-Bit Normal Word Addressing of Single Data in SIMD Mode

Figure 5-19 displays one possible SIMD mode, single data, Normal word addressed access. Figure 5-19 shows the data path for one transfer, but in this mode it is also important to note the pattern of iterative accesses. For Normal word accesses, the DSP accesses Normal words sequentially in memory. Table 5-11 on page 5-57 shows the pattern of SIMD mode Normal word accesses.

Table 5-11. Normal Word Addressing in SIMD Mode

Explicit Normal Word Accessed	Implicit Normal Word Accessed
Word X0 (Address LSB = 0)	Word X1 (Address LSB = 1)
Word X1 (Address LSB = 1)	Word X2 (Address LSB = 0)

For Normal word addressing, the DSP treats the data buses as two 32-bit Normal word lanes. The explicitly addressed (named in the instruction) 32-bit value transfers using the least significant Normal word lane of the PM or DM data bus. The implicitly addressed (not named in the instruction, but inferred from the address in SIMD mode) Normal word value transfers using the most significant Normal word lane of the PM or DM data bus. The DSP drives the other Normal word lanes of the data buses with zeros.

In Figure 5-19, the explicit access targets the named register `RX`, and the implicit access targets that register's complementary register `SX`. This case uses a PEx register with an `RX` mnemonic. If the syntax named a `PEy` register `SX` as the explicit target, the DSP would use that register's complement, `RX`, as the implicit target. For more information on complementary registers, see "Secondary Processing Element (PEy)" on page 2-36.

For Normal word accesses, the DSP zero-fills least significant 8 bits of the data register on loads and truncates these bits on stores to memory. For more information on arranging data in memory to take advantage of this access pattern, see "Arranging Data in Memory" on page 5-84.

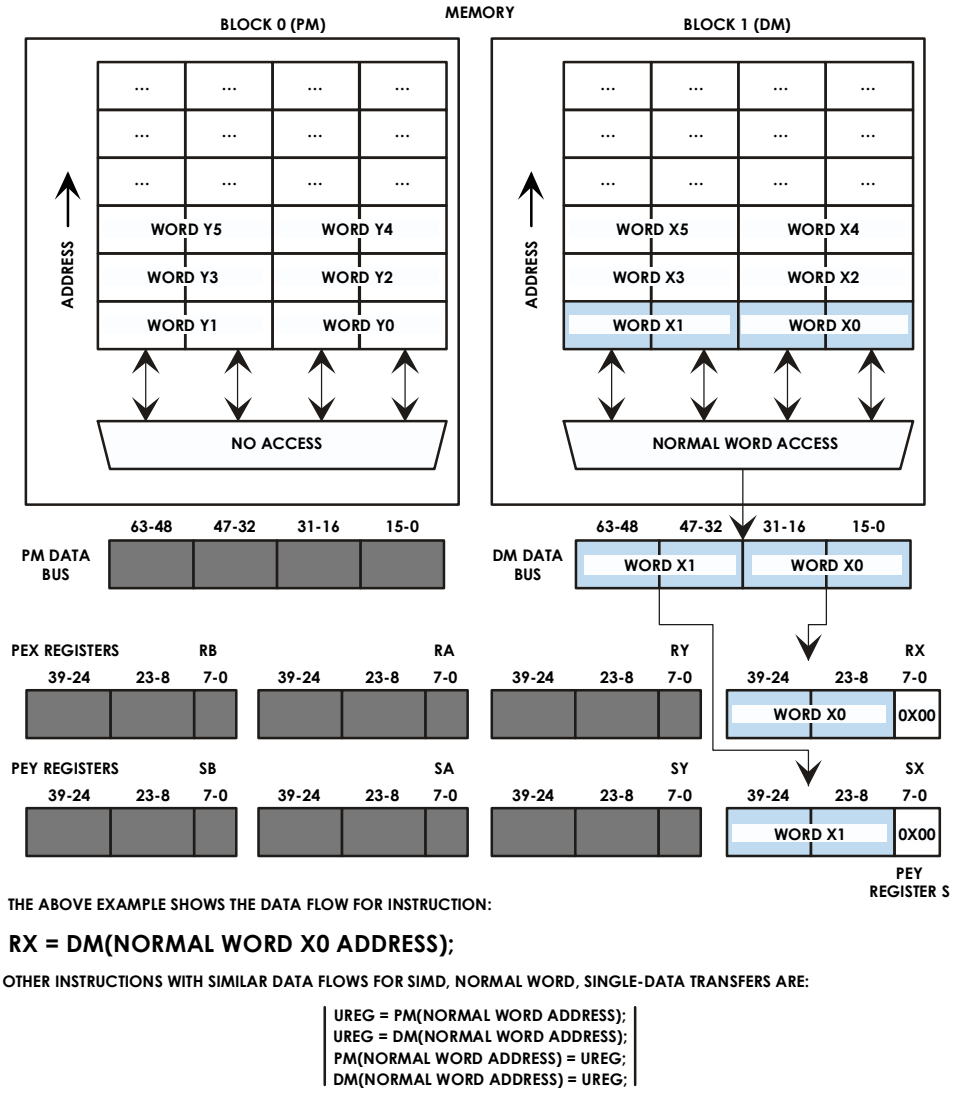


Figure 5-19. Normal Word Addressing of Single Data in SIMD Mode

32-Bit Normal Word Addressing of Dual Data in SISD Mode

Figure 5-20 displays one possible SISD mode, dual data, 32-bit Normal word addressed access.

For Normal word addressing, the DSP treats the data buses as two 32-bit Normal word lanes. The 32-bit values for Normal word accesses transfer using the least significant Normal word lanes of the PM and DM data buses. The DSP drives the other Normal word lanes of the data buses with zeros. Note that the accesses on both buses do not have to be the same word width. SISD mode dual-data accesses can handle any combination of Short word, Normal word, extended precision Normal word, or Long word accesses. For more information, see "Mixed Word Width Addressing of Dual Data in SISD Mode" on page 5-72.

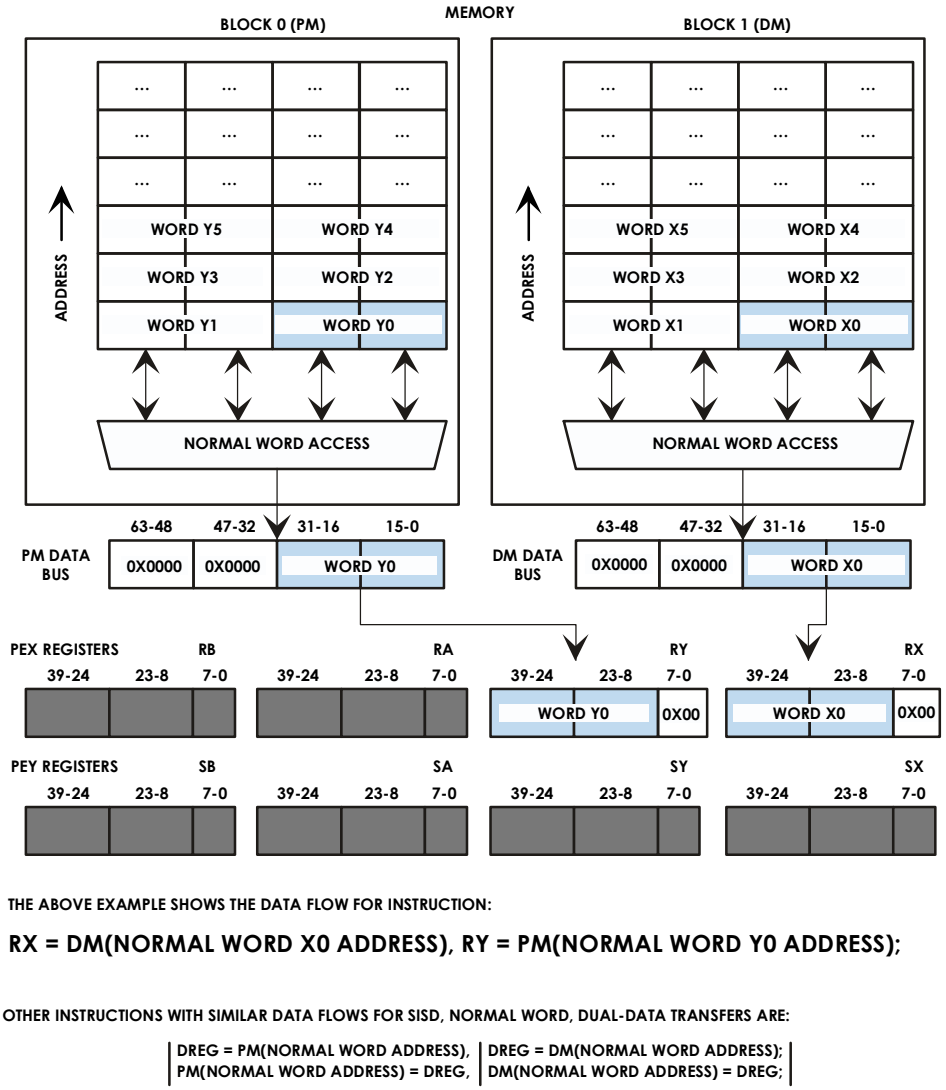
In Figure 5-20, the access targets PEx registers in a SISD mode operation. This case accesses `WORD X0` in block 1 and `WORD Y0` in block 0. Each of these words has a Normal word address with "0" for its least significant address bit. Other accesses within these 4-column locations have the addresses with the least significant bit of "1" and select `WORD X/Y1` from memory. The syntax targets registers `RX` and `RY` in PEx. The example would target PEy registers if using the syntax `SX` or `SY`.

For Normal word accesses, the DSP zero-fills least significant 8 bits of the data register on loads and truncates these bits on stores to memory.

32-Bit Normal Word Addressing of Dual Data in SIMD Mode

Figure 5-21 displays one possible SIMD mode, dual data, 32-bit Normal word addressed access.

For Normal word addressing, the DSP treats the data buses as two 32-bit Normal word lanes. The explicitly addressed (named in the instruction) 32-bit values transfer using the least significant Normal word lane of the PM or DM data bus. The implicitly addressed (not named in the instruction, but inferred from the address in SIMD mode) Normal word values



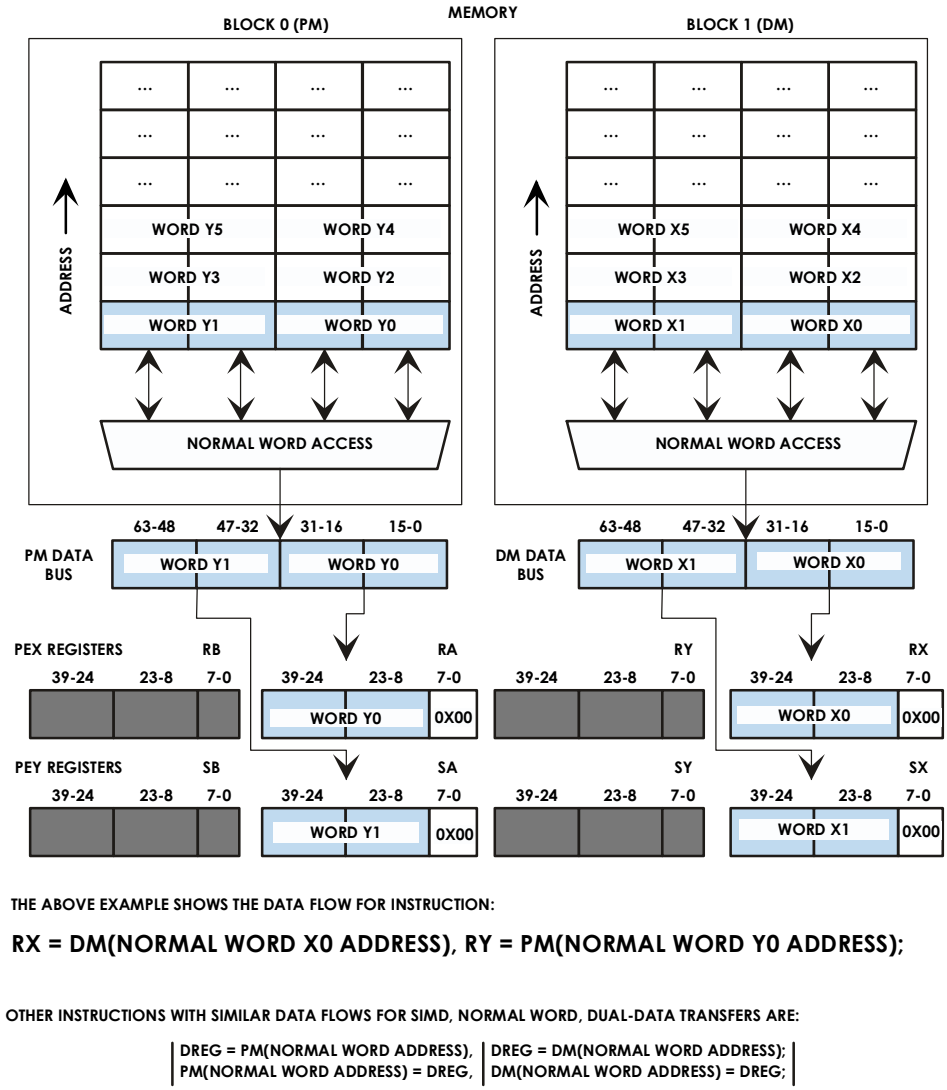
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(NORMAL WORD X0 ADDRESS), RY = PM(NORMAL WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, NORMAL WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(NORMAL WORD ADDRESS), PM(NORMAL WORD ADDRESS) = DREG,	DREG = DM(NORMAL WORD ADDRESS); DM(NORMAL WORD ADDRESS) = DREG;

Figure 5-20. Normal Word Addressing of Dual Data in SISD Mode



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(NORMAL WORD X0 ADDRESS), RY = PM(NORMAL WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, NORMAL WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(NORMAL WORD ADDRESS), PM(NORMAL WORD ADDRESS) = DREG,	DREG = DM(NORMAL WORD ADDRESS); DM(NORMAL WORD ADDRESS) = DREG;

Figure 5-21. Normal Word Addressing of Dual Data in SIMD Mode

transfer using the most significant Normal word lanes of the PM and DM data bus. The DSP drives the other Normal word lanes of the PM and DM data buses with zeros. Note that the accesses on both buses do not have to be the same word width. SIMD mode dual-data accesses can handle combinations of Short word and Normal word or extended precision Normal word and Long word accesses. For more information, see "Mixed Word Width Addressing of Dual Data in SIMD Mode" on page 5-74.

In Figure 5-21, the explicit access targets the named registers RX and RA, and the implicit access targets those register's complementary registers SX and SA. This case uses a PEx registers with the RX and RA mnemonics. If the syntax named PEy registers SX and SA as the explicit targets, the DSP would use those registers' complements RX and RA as the implicit targets. For more information on complementary registers, see "Secondary Processing Element (PEy)" on page 2-36.

For Normal word accesses, the DSP zero-fills least significant 8 bits of the data register on loads and truncates these bits on stores to memory.

Figure 5-21 shows the data path for one transfer, but in this mode it is also important to note the pattern of iterative accesses. For Normal word accesses, the DSP accesses Normal words sequentially in memory. Table 5-11 on page 5-24 shows the pattern of SIMD mode Normal word accesses. For more information on arranging data in memory to take advantage of this access pattern, see "Arranging Data in Memory" on page 5-84.

Extended Precision Normal Word Addressing of Single Data

Figure 5-22 displays one possible single data, 40-bit extended precision Normal word addressed access. For extended precision Normal word addressing, the DSP treats each data bus as a 40-bit extended precision Normal word lane. The 40-bit value for the extended precision Normal word access transfers using the most significant 40 bits of the PM or DM data bus. The DSP drives the lower 24 bits of the data buses with zeros.

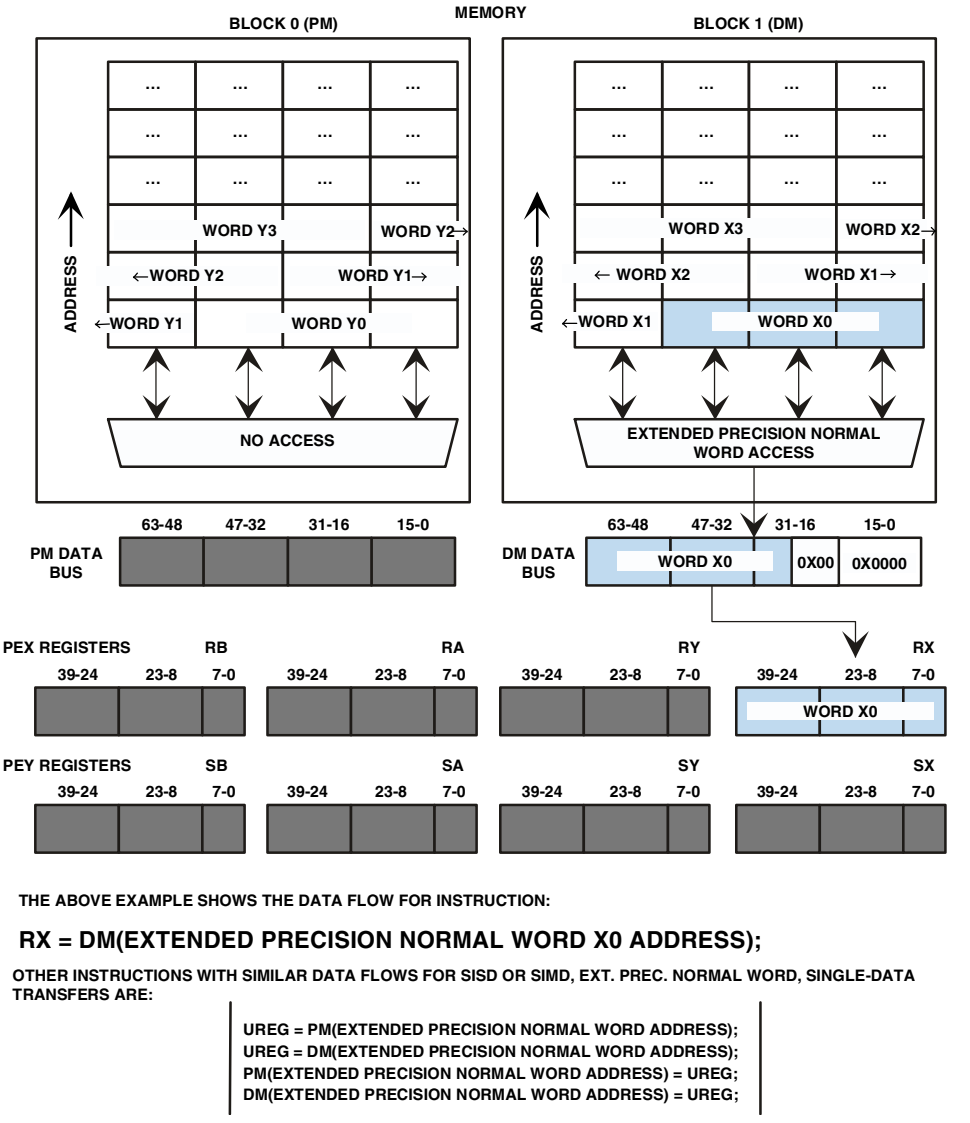


Figure 5-22. Extended Precision Normal Word Addressing of Single Data

In Figure 5-22, the access targets a PEx register in a SISD or SIMD mode operation; extended precision Normal word single-data access operate the same in SISD or SIMD mode. This case accesses `WORD X0` with syntax that targets register `RX` in PEx. The example would target a PEy register if using the syntax `SX`.

Extended Precision Normal Word Addressing of Dual Data in SISD Mode

Figure 5-23 displays one possible SISD mode, dual data, 40-bit extended precision normal word addressed access.

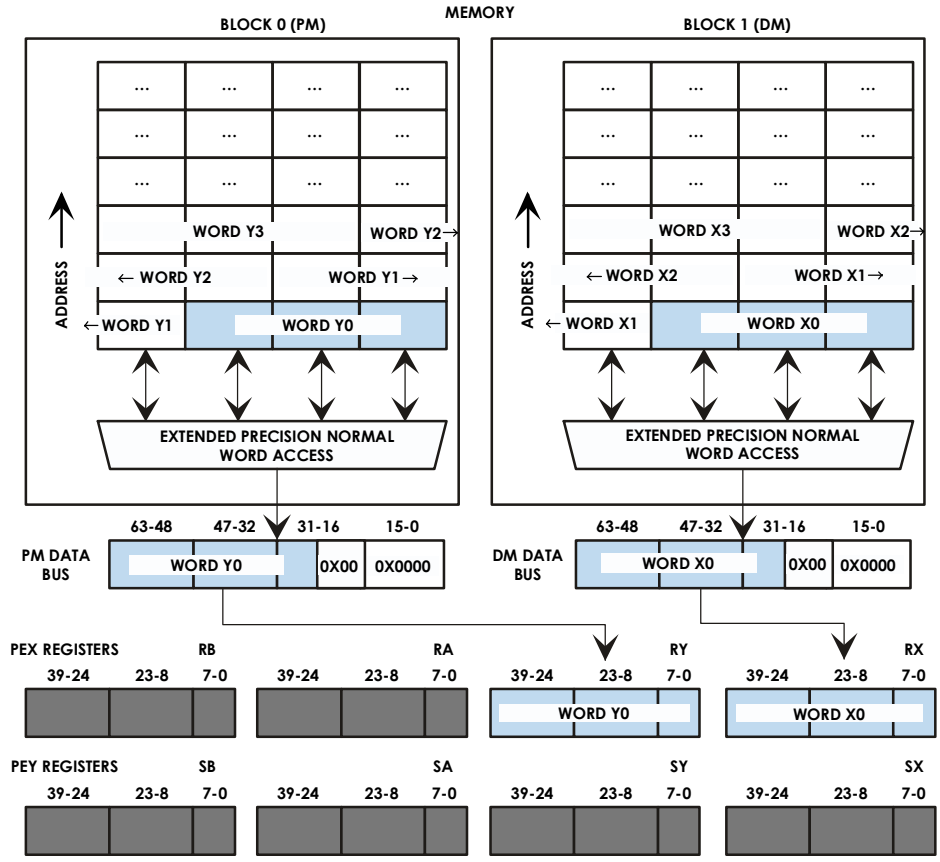
For extended precision Normal word addressing, the DSP treats each data bus as a 40-bit extended precision Normal word lane. The 40-bit values for the extended precision Normal word accesses transfer using the most significant 40 bits of the PM and DM data bus. The DSP drives the lower 24 bits of the data buses with zeros. Note that the accesses on both buses do not have to be the same word width. SISD mode dual-data accesses can handle any combination of Short word, Normal word, extended precision Normal word, or Long word accesses. For more information, see “Mixed Word Width Addressing of Dual Data in SISD Mode” on page 5-72.

In Figure 5-23, the access targets PEx registers in a SISD mode operation. This case accesses `WORD X0` in block 1 and `WORD Y0` in block 0 with syntax that targets registers `RX` and `RY` in PEx. The example would target a PEy registers if using the syntax `SX` or `SY`.

Extended Precision Normal Word Addressing of Dual Data in SIMD Mode

Figure 5-24 displays one possible SIMD mode, dual data, 40-bit extended precision normal word addressed access.

For extended precision Normal word addressing, the DSP treats each data bus as a 40-bit extended precision Normal word lane.



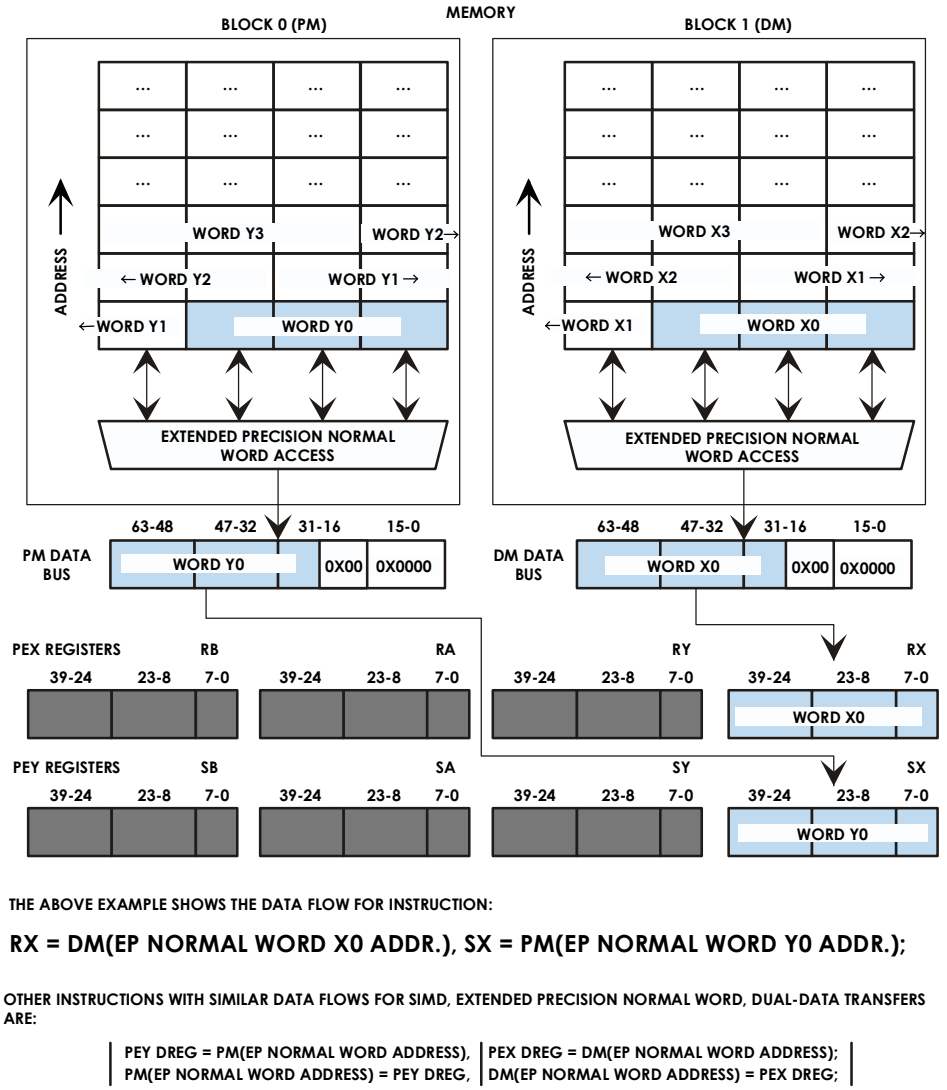
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(EP NORMAL WORD X0 ADDR.), RY = PM(EP NORMAL WORD Y0 ADDR.);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, EXTENDED PRECISION NORMAL WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(EXT. PREC. NORMAL WORD ADDRESS), PM(EXT. PREC. NORMAL WORD ADDRESS) = DREG,	DREG = DM(EXT. PREC. NORMAL WORD ADDRESS); DM(EXT. PREC. NORMAL WORD ADDRESS) = DREG;

Figure 5-23. Extended Precision Normal Word Addressing of Dual Data in SISD Mode



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(EP NORMAL WORD X0 ADDR.), SX = PM(EP NORMAL WORD Y0 ADDR.);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, EXTENDED PRECISION NORMAL WORD, DUAL-DATA TRANSFERS ARE:

PEY DREG = PM(EP NORMAL WORD ADDRESS), PM(EP NORMAL WORD ADDRESS) = PEY DREG,	PEX DREG = DM(EP NORMAL WORD ADDRESS); DM(EP NORMAL WORD ADDRESS) = PEX DREG;

Figure 5-24. Extended Precision Normal Word Addressing of Dual Data in SIMD Mode

Because this word size approaches the limit of the data buses capacity, this SIMD mode transfer only moves the explicitly addressed locations and restricts data bus usage. The explicitly addressed (named in the instruction) 40-bit values transferred over the DM bus must source or sink a PEx data register, and the explicitly addressed (named in the instruction) 40-bit values transferred over the PM bus must source or sink a PEy data register; there are no implicit transfers in this mode. The 40-bit values for the extended precision Normal word accesses transfer using the most significant 40 bits of the PM and DM data bus. The DSP drives the lower 24 bits of the data buses with zeros.



The accesses on both buses do not have to be the same word width. This special case of SIMD mode dual-data accesses can handle any combination of extended precision Normal word or Long word accesses. For more information, see "Mixed Word Width Addressing of Dual Data in SIMD Mode" on page 5-74.

In Figure 5-24, the access targets PEx and PEy registers in a SIMD mode operation. This case accesses `WORD X0` in block 1 with syntax that targets register `RX` in PEx and accesses `WORD Y0` in block 0 with syntax that targets register `SX` in PEy.

Long Word Addressing of Single Data

Figure 5-25 displays one possible single data, Long word addressed access.

For Long word addressing, the DSP treats each data bus as a 64-bit Long word lane. The 64-bit value for the Long word access transfers using the full width of the PM or DM data bus.

In Figure 5-25, the access targets a PEx register in a SISD or SIMD mode operation; Long word single-data access operate the same in SISD or SIMD mode. This case accesses `WORD X0` with syntax that explicitly targets register `RX` and implicitly targets its neighbor register `RY` in PEx. The example would target PEy registers if using the syntax `SX`. For more infor-

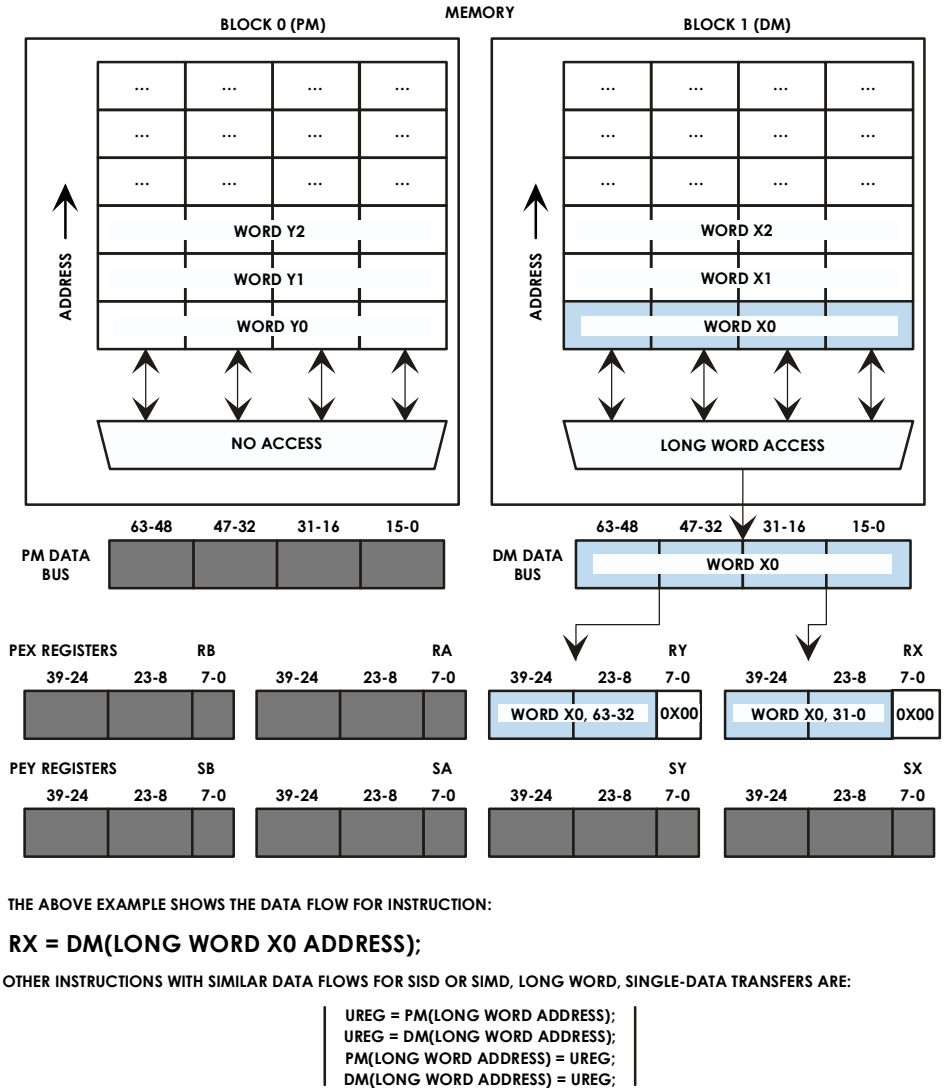


Figure 5-25. Long Word Addressing of Single Data

mation on how neighbor registers (listed in Table 5-7 on page 5-42) work, see "Long Word (64-Bit) Accesses" on page 5-41.

Long Word Addressing of Dual Data in SISD Mode

Figure 5-26 displays one possible SISD mode, dual data, long word addressed access.

For Long word addressing, the DSP treats each data bus as a 64-bit Long word lane. The 64-bit values for the Long word accesses transfer using the full width of the PM or DM data bus.

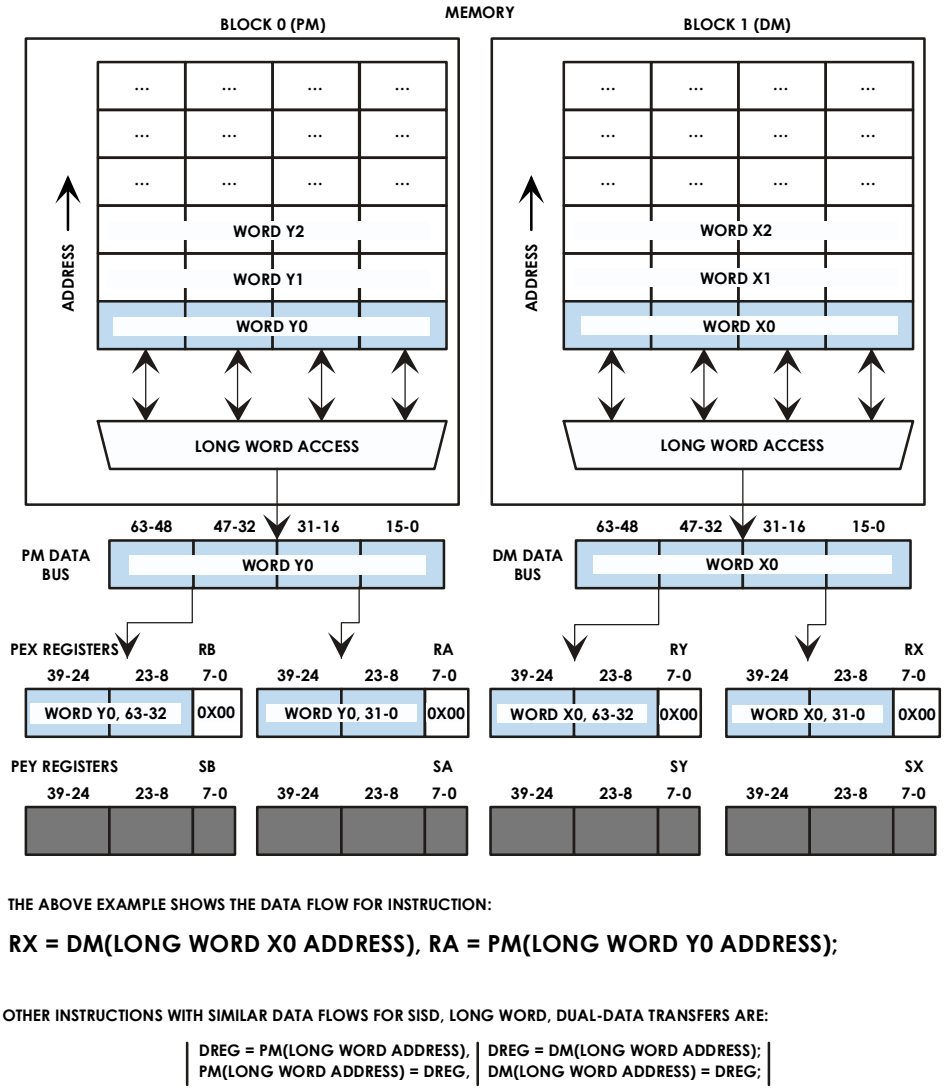
In Figure 5-26, the access targets PEx registers in a SISD mode operation. This case accesses `WORD XO` and `WORD YO` with syntax that explicitly targets registers `RX` registers `RA` and implicitly targets their neighbor registers `RY` and `RB` in PEx. The example would target PEy registers if using the syntax `SX` and `SA`. For more information on how neighbor registers (listed in Table 5-7 on page 5-42) work, see "Long Word (64-Bit) Accesses" on page 5-41.

Programs must be careful not to explicitly target neighbor registers in this case. While the syntax lets programs target these registers, one of the explicit accesses targets the other access's implicit target. The DSP resolves this conflict by performing only the access with higher priority. For more information on the priority order of data register file accesses, see "Data Register File" on page 2-29.

Long Word Addressing of Dual Data in SIMD Mode

Figure 5-27 displays one possible SIMD mode, dual data, Long word addressed access targeting internal memory space.

For Long word addressing, the DSP treats each data bus as a 64-bit Long word lane. The 64-bit values for the Long word accesses transfer using the full width of the PM or DM data bus.



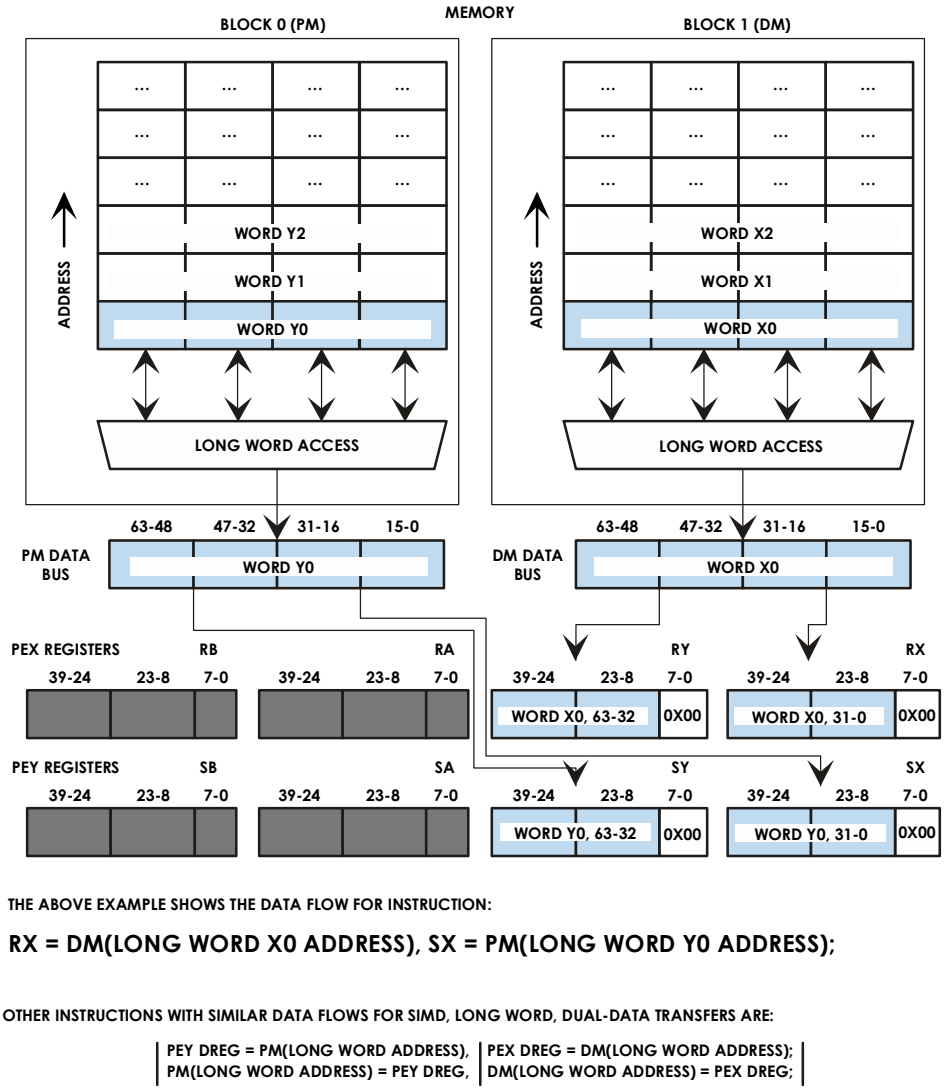
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(LONG WORD X0 ADDRESS), RA = PM(LONG WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, LONG WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(LONG WORD ADDRESS), PM(LONG WORD ADDRESS) = DREG,	DREG = DM(LONG WORD ADDRESS); DM(LONG WORD ADDRESS) = DREG;

Figure 5-26. Long Word Addressing of Dual Data in SISD Mode



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(LONG WORD X0 ADDRESS), SX = PM(LONG WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SIMD, LONG WORD, DUAL-DATA TRANSFERS ARE:

PEY DREG = PM(LONG WORD ADDRESS), PM(LONG WORD ADDRESS) = PEY DREG,	PEX DREG = DM(LONG WORD ADDRESS); DM(LONG WORD ADDRESS) = PEX DREG;

Figure 5-27. Long Word Addressing of Dual Data in SIMD Mode

Because this word size approaches the limit of the data buses capacity, this SIMD mode transfer only moves the explicitly addressed locations and restricts data bus usage. The explicitly addressed (named in the instruction) 64-bit values transferred over the DM bus must source or sink a PEx data register, and the explicitly addressed (named in the instruction) 64-bit values transferred over the PM bus must source or sink a PEy data register; there are no implicit transfers in this mode.

In Figure 5-27, the access targets PEx and PEy registers in a SIMD mode operation. This case accesses `WORD X0` in block 1 with syntax that targets register `RX` and its neighbor register `RY` in PEx and accesses `WORD Y0` in block 0 with syntax that targets register `SX` and its neighbor register `SY` in PEy. For more information on how neighbor registers (listed in "Neighbor Registers for Long Word Accesses" on page 5-42) work, see "Long Word (64-Bit) Accesses" on page 5-41.



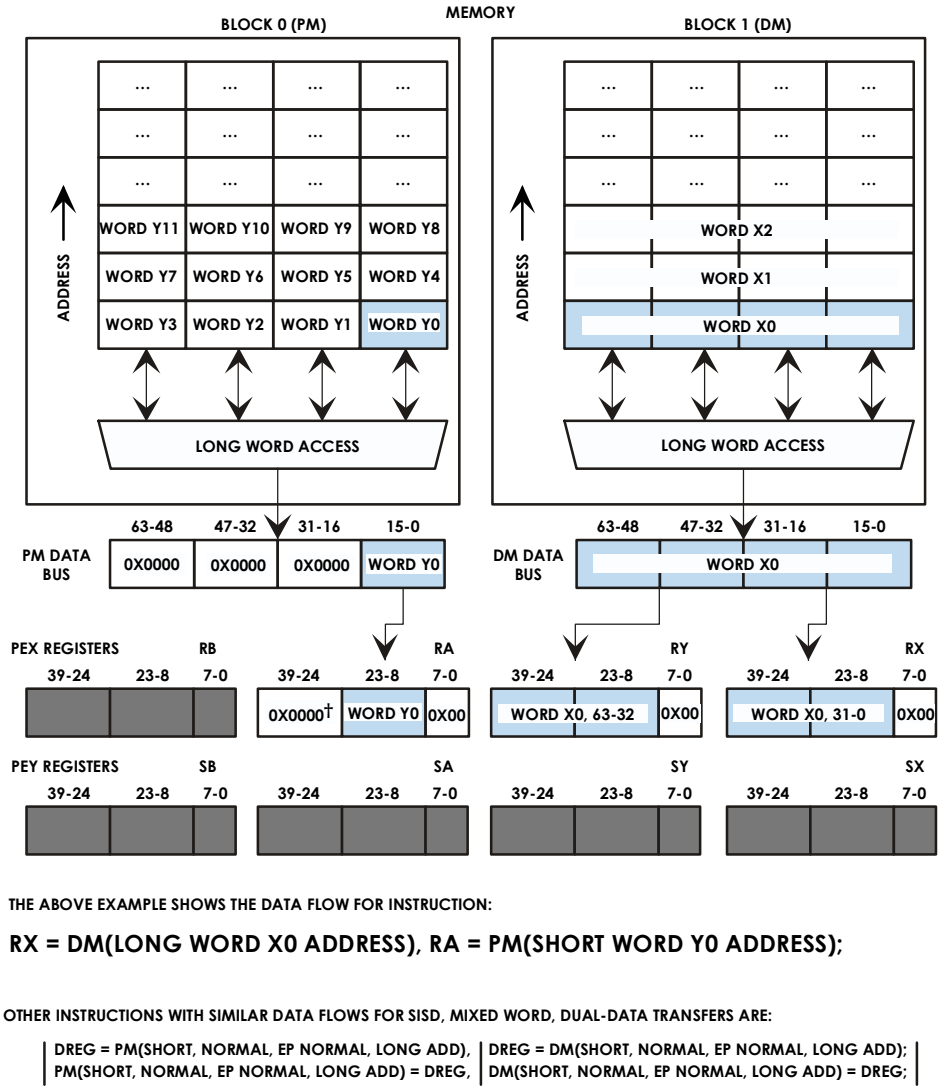
The accesses on both buses do not have to be the same word width. This special case of SIMD mode dual-data accesses can handle any combination of extended precision Normal word or Long word accesses. "Mixed Word Width Addressing of Dual Data in SIMD Mode" on page 5-74

Mixed Word Width Addressing of Dual Data in SISD Mode

Figure 5-28 displays an example of a mixed word width, dual data, SISD mode access.

This example shows how the DSP transfers a Long word access on the DM bus and transfers a Short word access on the PM bus. The memory architecture permits mixing all other combinations of dual-data SISD mode

Short word, Normal word, extended precision Normal word, and Long word accesses.



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(LONG WORD X0 ADDRESS), RA = PM(SHORT WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR SISD, MIXED WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(SHORT, NORMAL, EP NORMAL, LONG ADD), PM(SHORT, NORMAL, EP NORMAL, LONG ADD) = DREG,	DREG = DM(SHORT, NORMAL, EP NORMAL, LONG ADD); DM(SHORT, NORMAL, EP NORMAL, LONG ADD) = DREG;

Figure 5-28. Mixed Word Width Addressing of Dual Data in SISD Mode



In case of conflicting dual access to the data register file, the DSP only performs the access with higher priority. For more information on how the DSP prioritizes accesses, see "Data Register File" on page 2-29.

Mixed Word Width Addressing of Dual Data in SIMD Mode

Figure 5-29 displays an example of a mixed word width, dual data, SIMD mode access.

This example shows how the DSP transfers a Long word access on the DM bus and transfers an extended precision Normal word access on the PM bus.



The memory architecture permits mixing SIMD mode dual data Short word and Normal word accesses or extended precision Normal word and Long word accesses. No other combinations of mixed word dual-data SIMD mode accesses are permissible.

Broadcast Load Access

Figure 5-30, Figure 5-31 on page 5-77, Figure 5-32 on page 5-78, Figure 5-33 on page 5-79, Figure 5-34 on page 5-80, Figure 5-35 on page 5-81, Figure 5-36 on page 5-82, and Figure 5-37 on page 5-83 provide examples of broadcast load accesses for single- and dual-data transfers.

These examples show that the broadcast load's memory and register access is a hybrid of the corresponding non-broadcast SISD and SIMD mode accesses. The exceptions to this relation are broadcast load dual-data, extended precision Normal word and Long word accesses. These broadcast accesses differ from their corresponding non-broadcast mode accesses.

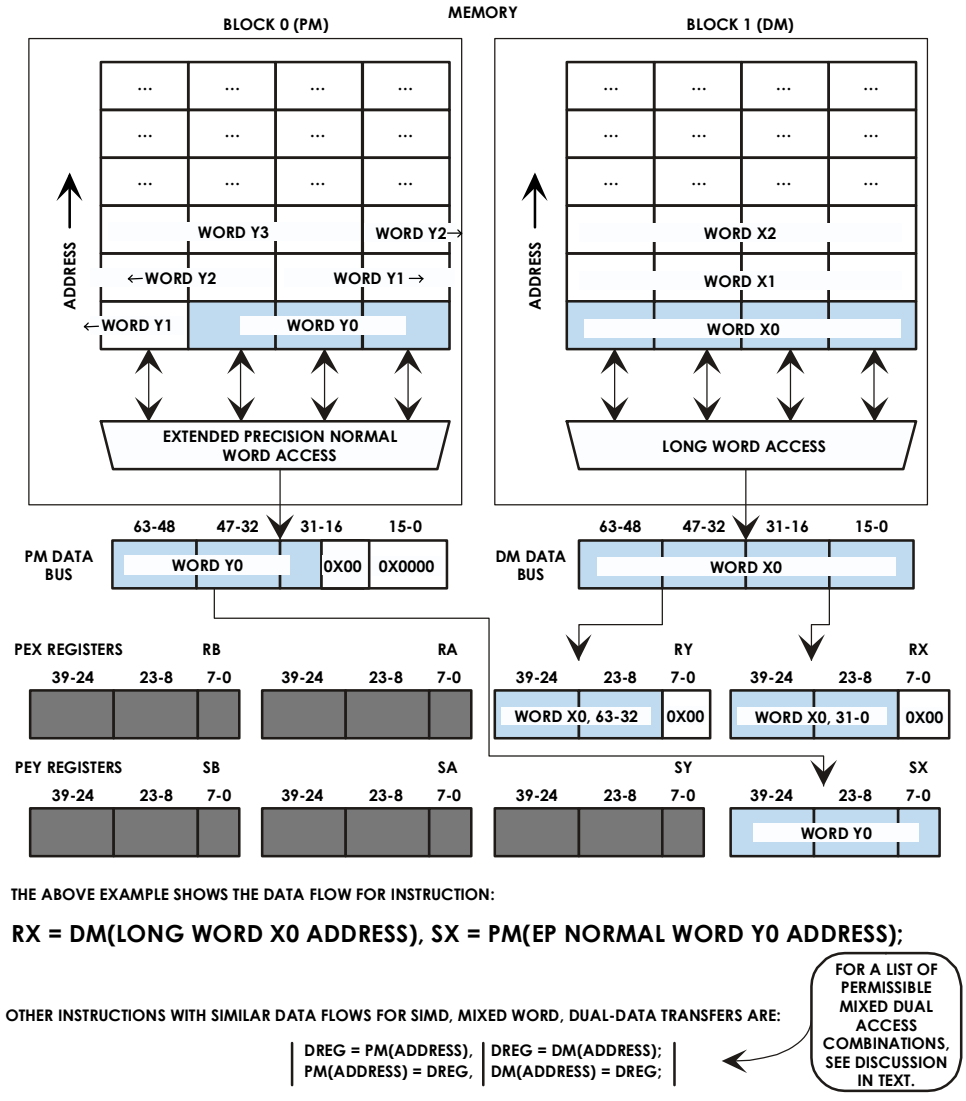


Figure 5-29. Mixed Word Width Addressing of Dual Data in SIMD Mode

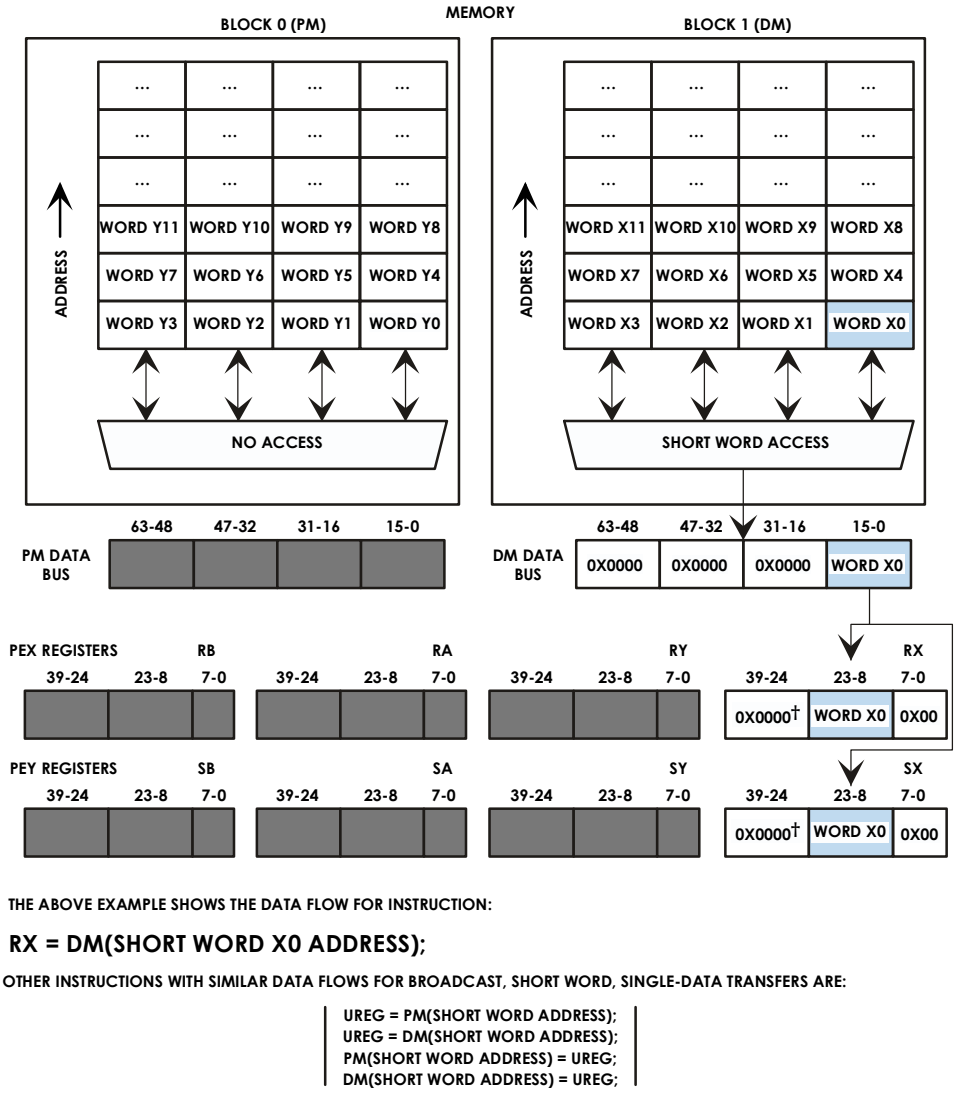
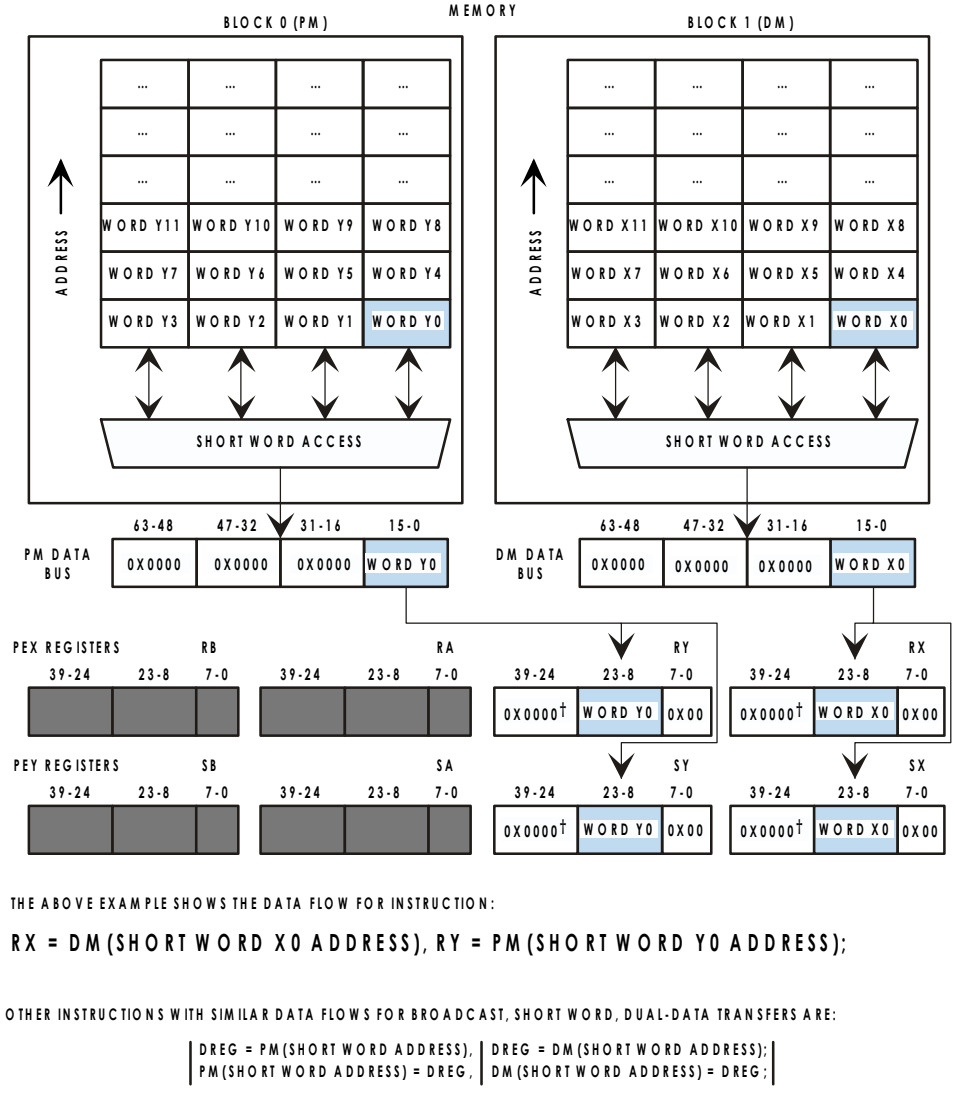


Figure 5-30. Short Word Addressing of Single Data in Broadcast Load



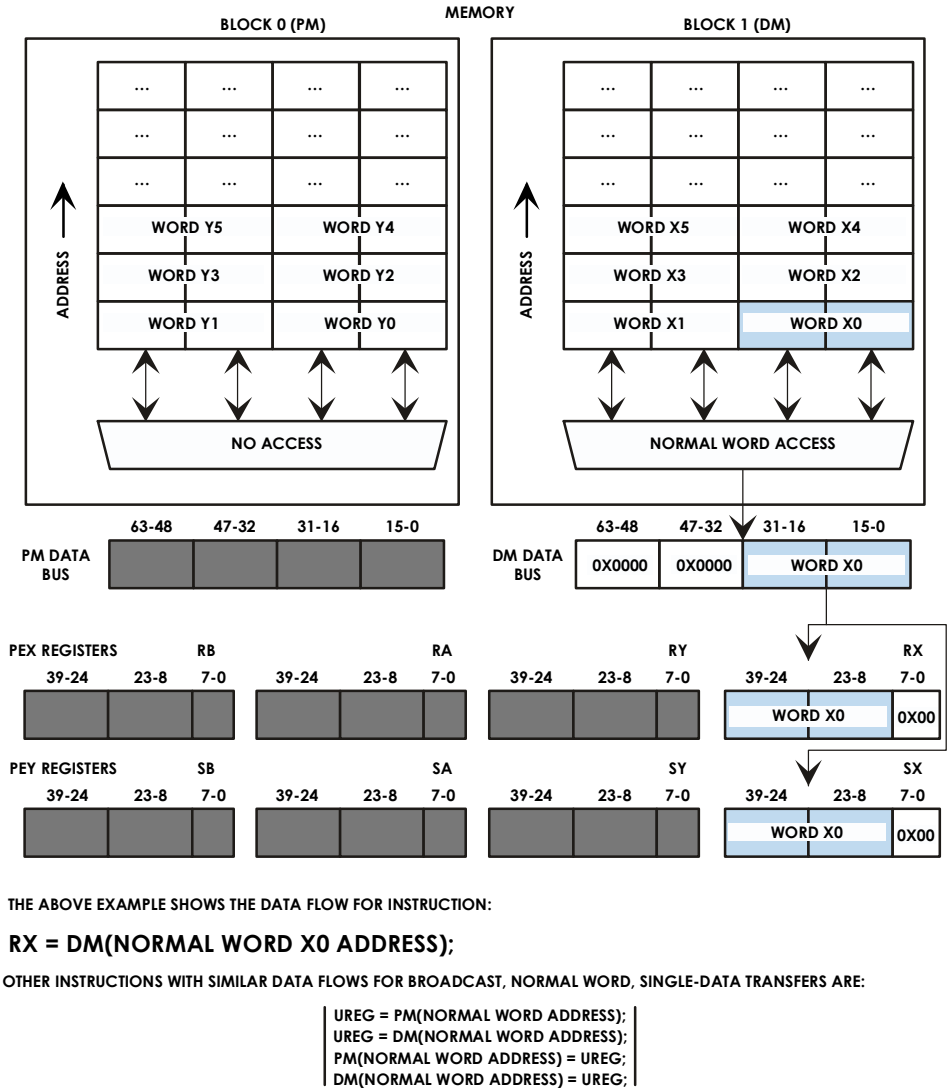
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(SHORT WORD X0 ADDRESS), RY = PM(SHORT WORD Y0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR BROADCAST, SHORT WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(SHORT WORD ADDRESS), PM(SHORT WORD ADDRESS) = DREG,	DREG = DM(SHORT WORD ADDRESS); DM(SHORT WORD ADDRESS) = DREG;

Figure 5-31. Short Word Addressing of Dual Data in Broadcast Load



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(NORMAL WORD X0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR BROADCAST, NORMAL WORD, SINGLE-DATA TRANSFERS ARE:

```
UREG = PM(NORMAL WORD ADDRESS);
UREG = DM(NORMAL WORD ADDRESS);
PM(NORMAL WORD ADDRESS) = UREG;
DM(NORMAL WORD ADDRESS) = UREG;
```

Figure 5-32. Normal Word Addressing of Single Data in Broadcast Load

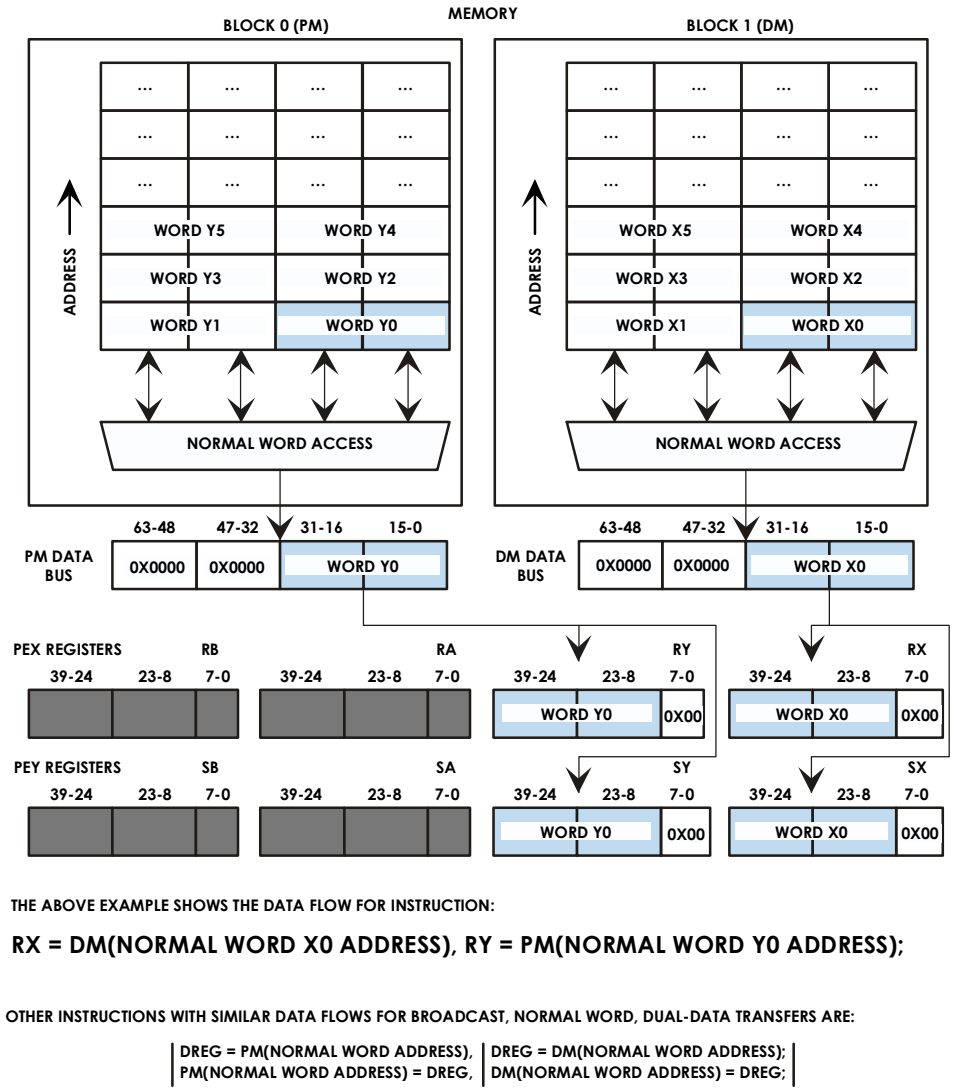
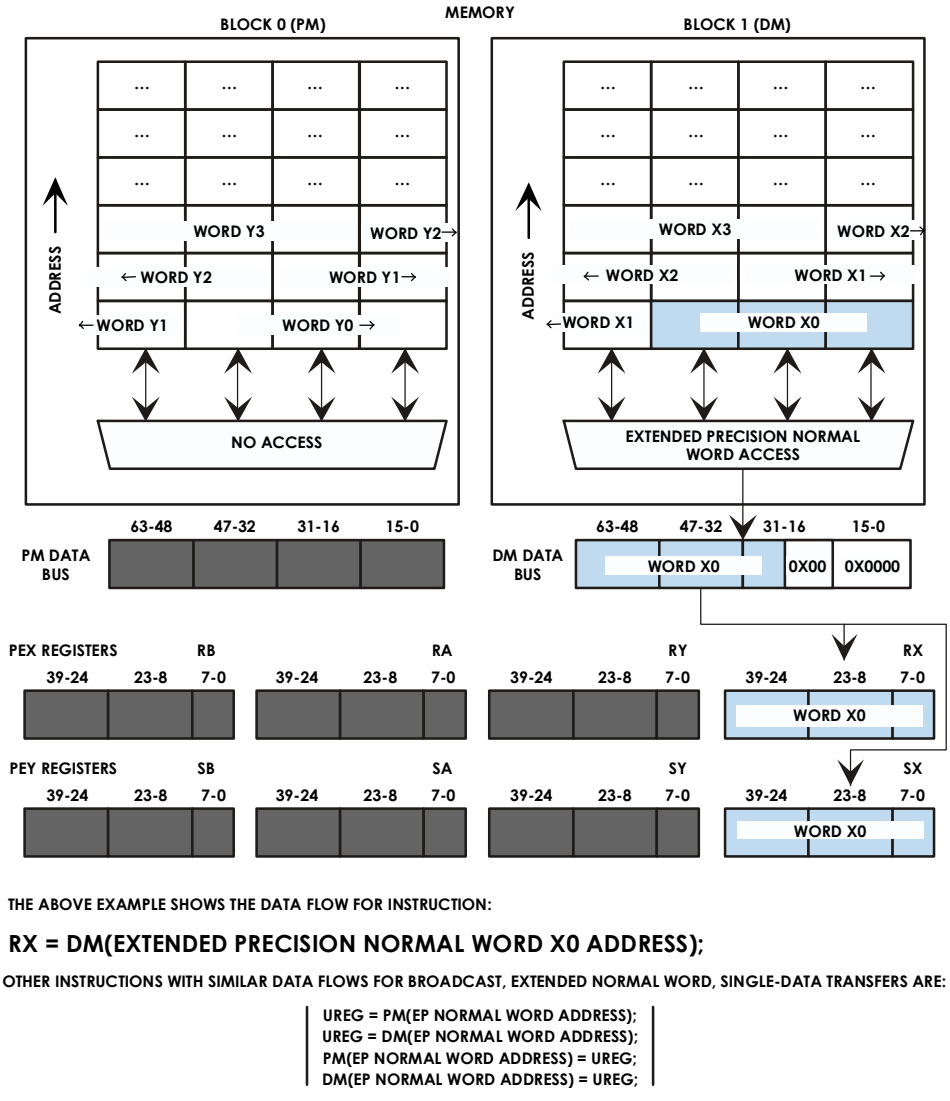


Figure 5-33. Normal Word Addressing of Dual Data in Broadcast Load



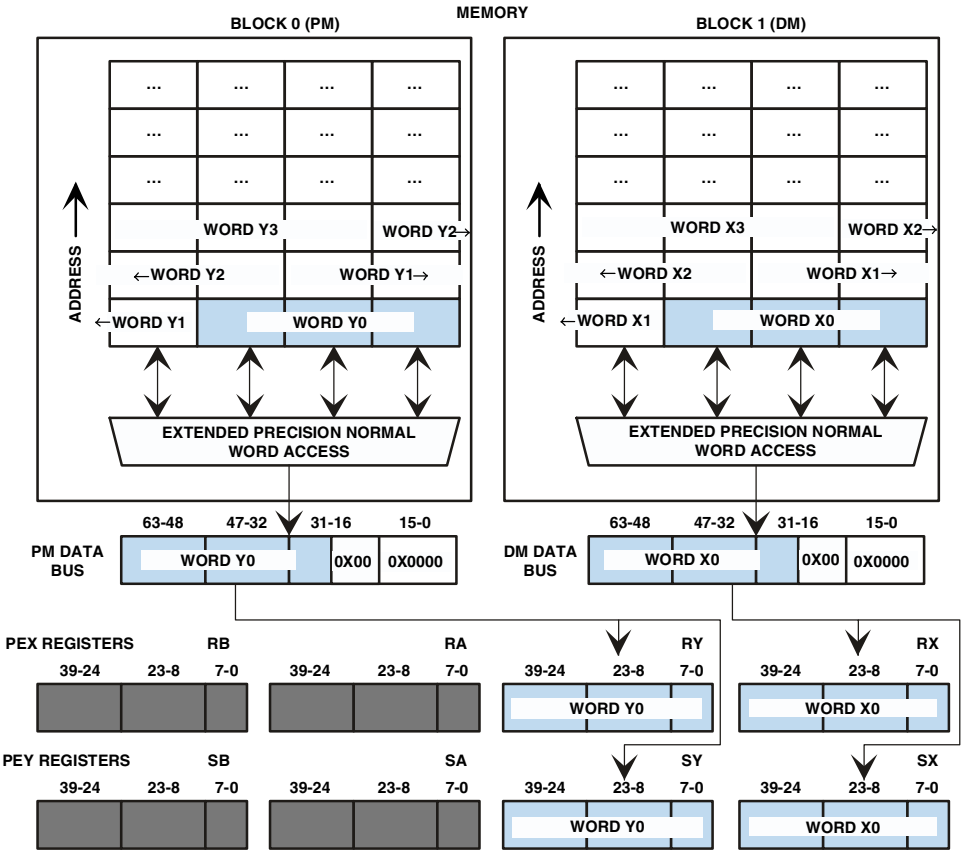
THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(EXTENDED PRECISION NORMAL WORD X0 ADDRESS);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR BROADCAST, EXTENDED NORMAL WORD, SINGLE-DATA TRANSFERS ARE:

```
UREG = PM(EP NORMAL WORD ADDRESS);
UREG = DM(EP NORMAL WORD ADDRESS);
PM(EP NORMAL WORD ADDRESS) = UREG;
DM(EP NORMAL WORD ADDRESS) = UREG;
```

Figure 5-34. Extended Precision Normal Word Addressing of Single Data in Broadcast Load



THE ABOVE EXAMPLE SHOWS THE DATA FLOW FOR INSTRUCTION:

RX = DM(EP NORMAL WORD X0 ADDR.), RY = PM(EP NORMAL WORD Y0 ADDR.);

OTHER INSTRUCTIONS WITH SIMILAR DATA FLOWS FOR BROADCAST, EXTENDED NORMAL WORD, DUAL-DATA TRANSFERS ARE:

DREG = PM(EP NORMAL WORD ADDRESS), PM(EP NORMAL WORD ADDRESS) = DREG,	DREG = DM(EPNORMAL WORD ADDRESS); DM(EP NORMAL WORD ADDRESS) = DREG;

Figure 5-35. Extended Precision Normal Word Addressing of Dual Data in Broadcast Load

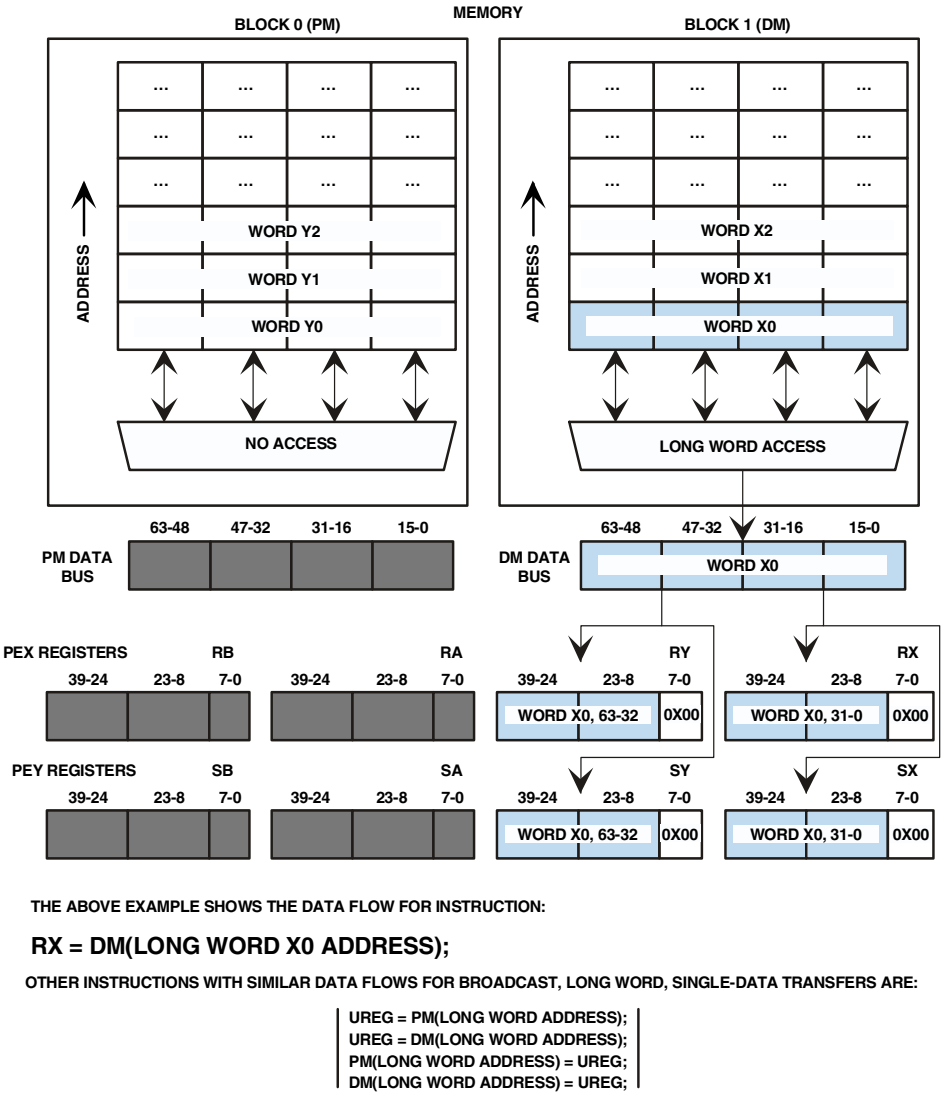


Figure 5-36. Long Word Addressing of Single Data in Broadcast Load

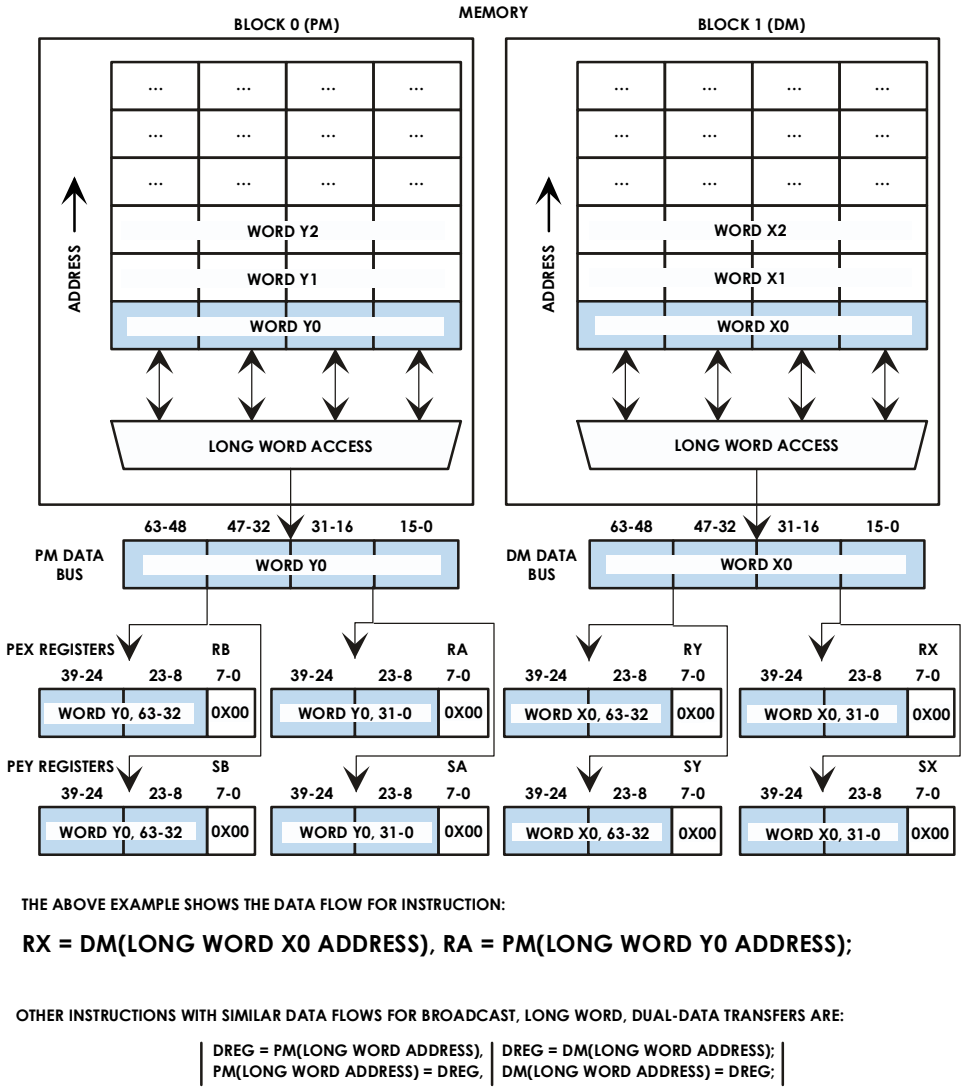


Figure 5-37. Long Word Addressing of Dual Data in Broadcast Load

Arranging Data in Memory

Each DSP's access to internal memory gets data from either a 4-columns (Long, Normal, or Short word) or 3-columns (instruction or extended precision Normal word) memory location. For more information on how the DSP accesses 4- or 3-column data, see "Memory Organization and Word Size" on page 5-22.

To take advantage of the DSP's data accesses to 4- and 3-column locations, programs must adjust the interleaving of data into memory locations to accommodate the memory access mode. The following guidelines provide an overview of how programs should interleave data in memory locations. For more information and examples, see the *ADSP-21160 SHARC DSP Instruction Set Reference*:

- Programs can use odd or even modify values (1, 2, 3, …) to step through a buffer in **single-or dual-data, SISD or Broadcast load mode** regardless of the data word size (Long word, extended precision Normal word, Normal word, or Short word).
- Programs should use multiple of 4 modify values (4, 8, 12, …) to step through a buffer of **Short word data in single-or dual-data, SIMD mode.**
- Programs should use multiple of 2 modify values (2, 4, 6, …) to step through a buffer of **Normal word data in single- or dual-data SIMD mode.**
- Programs can use odd or even modify values (1, 2, 3, …) to step through a buffer of **Long word or extended precision Normal word data in single- or dual-data, SIMD mode**.

6 I/O PROCESSOR

The DSP's I/O processor manages Direct Memory Accessing (DMA) of DSP memory through the external, link, and serial ports. Each DMA operation transfers an entire block of data. By managing DMA, the I/O processor lets programs move data as a background task while using the processor core for other DSP operations.

Overview

The I/O processor's architecture, which appears in Figure 6-1 on page 6-3, supports a number of DMA operations. These operations include the following transfer types:

- Internal memory ↔ external memory or external peripherals
- Internal memory ↔ internal memory of other DSPs
- Internal memory ↔ host processor
- Internal memory ↔ serial port I/O
- Internal memory ↔ link port I/O
- External memory ↔ external peripherals

This chapter describes the I/O processor and how the I/O processor controls external port, link port, and serial port operations. For information on connecting external devices to the external port, link ports, or serial ports, see "External Port", "Link Ports", or "Serial Ports".

DMA transfers between internal memory and external memory, multiprocessor memory, or a host use the DSP's external port. For these types of transfers, a program sets up the DMA controller with the internal memory buffer size and address, the address modifier, and the direction of transfer. These DMA set up parameters are the Transfer Control Block (TCB) for the DMA transfer. After setup, the DMA transfers begins when the program enables the channel and continues until the I/O processor transfers the entire buffer to or from DSP memory.

Similarly, DMA transfers between internal memory and link or serial ports have DMA parameters (a TCB). When the I/O processor performs DMA between internal memory and one of these ports, the program sets up the parameters and the I/O goes through the port instead of the external bus.

The direction (receive or transmit) of the I/O port determines the direction of data transfer. When the port receives data, the I/O processor automatically transfers the data to internal memory. When the port needs to transmit a word, the I/O processor automatically fetches the data from internal memory.

The I/O processor also lets the DSP system perform DMA transfers between an external device and external memory. This external to external transfer only uses the external port and I/O processor.

External devices can control external port DMA transfers in two ways. If the external device can handle bus mastership, the external device can master reads or writes to DMA buffers on the DSP. External devices also can assert a DMA Request input ($\overline{\text{DMARx}}$) to request service.

To further minimize loading on the processor core, the I/O processor supports chained DMA operations. When using chained DMA, a program can set up a DMA transfer to automatically set up and start the next DMA transfer after the current one completes.

Figure 6-1 shows the DSP's I/O processor, related ports, and buses. Figure 6-8 on page 6-68 shows more detail on DMA channel data paths.

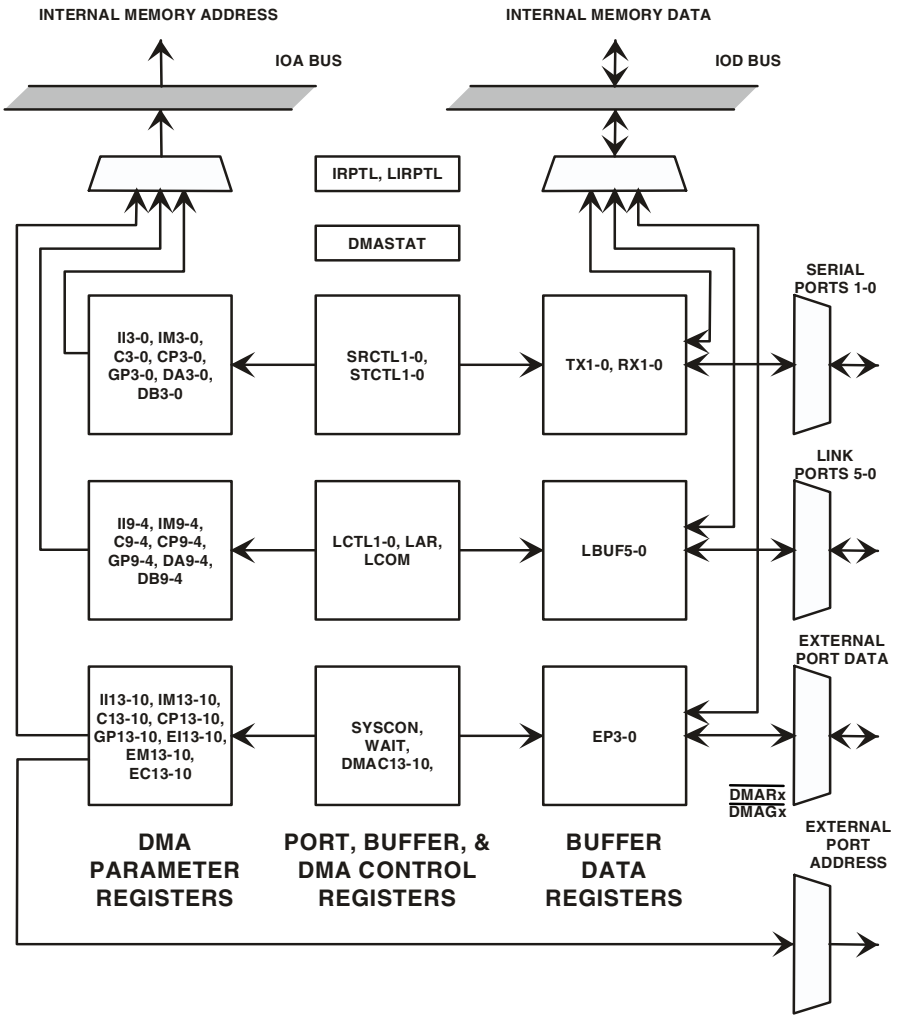


Figure 6-1. I/O Processor Block Diagram

The Data Buffer Registers column in Figure 6-1 shows the data buffer registers for each port. These registers include:

- **External Port Buffer registers** (EBPx). These 64-bit buffers for the external port have eight-position FIFOs for transmitting or receiving data when interfacing with a host or external devices such as memory and memory mapped devices.
- **Link Port Buffer registers** (LBUFx). These buffers for the link ports have two-position FIFOs for transmitting or receiving DMA data when connected to another link port.
- **Serial Port Receive Buffer registers** (RXx). These receive buffers for the serial ports have two-position FIFOs for receiving data when connected to another serial device.
- **Serial Port Transmit Buffer registers** (TXx). These transmit buffers for the serial ports have two position FIFOs for transmitting data when connected to another serial device.

The Port, Buffer, and DMA Control Registers column in Figure 6-1 on page 6-3 shows the control registers for the ports and DMA channels. These registers include:

- **System Configuration register** (SYSCON). This register configures packing, priority, and word order for the external port.
- **Waitstate and Access Mode register** (WAIT). This register configures handshake, idle cycle insertion, and waitstate insertion for external memory DMA accesses.
- **External Port DMA Control registers** (DMACx). These control registers for each external port DMA channel select the direction, format, and handshake and enable chaining, transfer mode, and DMA start.
- **Link Port Common Controls register** (LCOM). This register indicates link buffer packing and error status for link port operations.

- **Link Port Assignment register** (LAR). This register assigns link buffers to link ports for link port operations.
- **Link Port Control registers** (LCTLx). These control registers (each register controls three link buffers) select the direction, word width, and transfer rate and enable chaining, 2-D DMA mode, and DMA start.
- **Serial Port Receive Control registers** (SRCTLx). These control registers for each port select the receive format; monitor FIFO status; and enable chaining, 2-D DMA mode, and DMA start.
- **Serial Port Transmit Control registers** (STCTLx). These control registers for each port select the transmit format; monitor FIFO status; and enable chaining, 2-D DMA mode, and DMA start.

The DMA Parameter Registers column in Figure 6-2 on page 6-7 shows the parameter registers for each DMA channel. These registers function similarly to data address generator registers and include:

- **Internal Index registers** (IIx). An index register provides an internal memory address, acting as a pointer to the next internal memory DMA read or write location.
- **Internal Modify registers** (IMx). A modify register provides the signed increment by which the DMA controller post-modifies the corresponding internal memory index register after the DMA read or write.
- **Count registers** (Cx). A count register indicates the number of words remaining to be transferred to or from internal memory on the corresponding DMA channel.

- **Chain Pointer registers** (CPx). A chain pointer register holds the starting address of the Transfer Control Block (parameter register values) for the next DMA operation on the corresponding channel. These registers also control whether the I/O processor generates an interrupt when the current DMA process ends.
- **General Purpose registers** (GPx). A general purpose DMA register holds an address or other value.
- **Dimension A and B registers** (DAx **and** DBx). Dimension registers hold the counts for the A and B dimensions of a 2-dimensional DMA. For more information on two-dimensional DMA, see "Using Two-Dimensional Link Port DMA" on page 6-83 or "Using Two-Dimensional Serial Port DMA" on page 6-93.
- **External Index registers** (EIx). An index register provides an external memory address, acting as a pointer to the next external memory DMA read or write location.
- **External Modify registers** (EMx). A modify register provides the increment by which the DMA controller post-modifies the corresponding external memory index register after the DMA read or write.
- **External Count registers** (ECx). An external count register indicates the number of words remaining to be transferred to or from external memory on the corresponding DMA channel.

Figure 6-3 on page 6-8 shows a block diagram of the I/O processor's address generator (DMA controller). Table 6-1 on page 6-11 lists the parameter registers for each DMA channel. The parameter registers are uninitialized following a processor reset.

The I/O processor generates addresses for DMA channels much the same way that the Data Address Generators (DAGs) generate addresses for data memory accesses. Each channel has a set of parameter registers including an index register (IIx) and modify register (IMx) that the I/O processor

Register	Function	Width	Description
IIx	Internal Index Register	18-bits*	Address of buffer in internal memory
IMx	Internal Modify Register	16-bits	Stride for internal buffer
Cx	Internal Count Register	16-bits	Length of internal buffer
CPx	Chain Pointer Register	19-bits*	Chain pointer for DMA chaining
GPx	General Purpose Register	18-bits	User definable
DBx	Dimension B Register	16-bits	Count of dimension B buffer
DAx	Dimension A Register	16-bits	Count of dimension A buffer
EIx	External Index Register	32-bits	Address of buffer in external memory
EMx	External Modify Register	32-bits	Stride for external buffer
ECx	External Count Register	32-bits	Length of external buffer

External Port DMA channels only

* Offset by 0x40000 for internal addressing in normal word space

Figure 6-2. ADSP-21160 DSP's DMA Parameter Register

uses to address a data buffer in internal memory. The index register must be initialized with a starting address for the data buffer. As part of the DMA operation, the I/O processor outputs the address in the index register onto the DSP's IO (I/O Address) bus and applies the address to internal memory during each DMA cycle—a clock cycle in which a DMA transfer is taking place.)

All addresses in the index (`IIx`) registers are offset by a value matching the DSP's first internal Normal word addressed RAM location, before the I/O processor uses the addresses. For the ADSP-21160, this offset value is 0x0004 0000.

While DMA addresses must always be Normal word (32-bit) memory, the internal memory data transfer sizes may be 64-, 48-, or 32-bits. External memory data transfer sizes may be 64-, 32-, or 16-bits. The I/O processor can transfer Short word data (16-bit) using the packing capability of the external port and serial port DMA channels.

After transferring each data word to or from internal memory, the I/O processor adds the modify value to the index register to generate the address for the next DMA transfer and writes the modified index value to

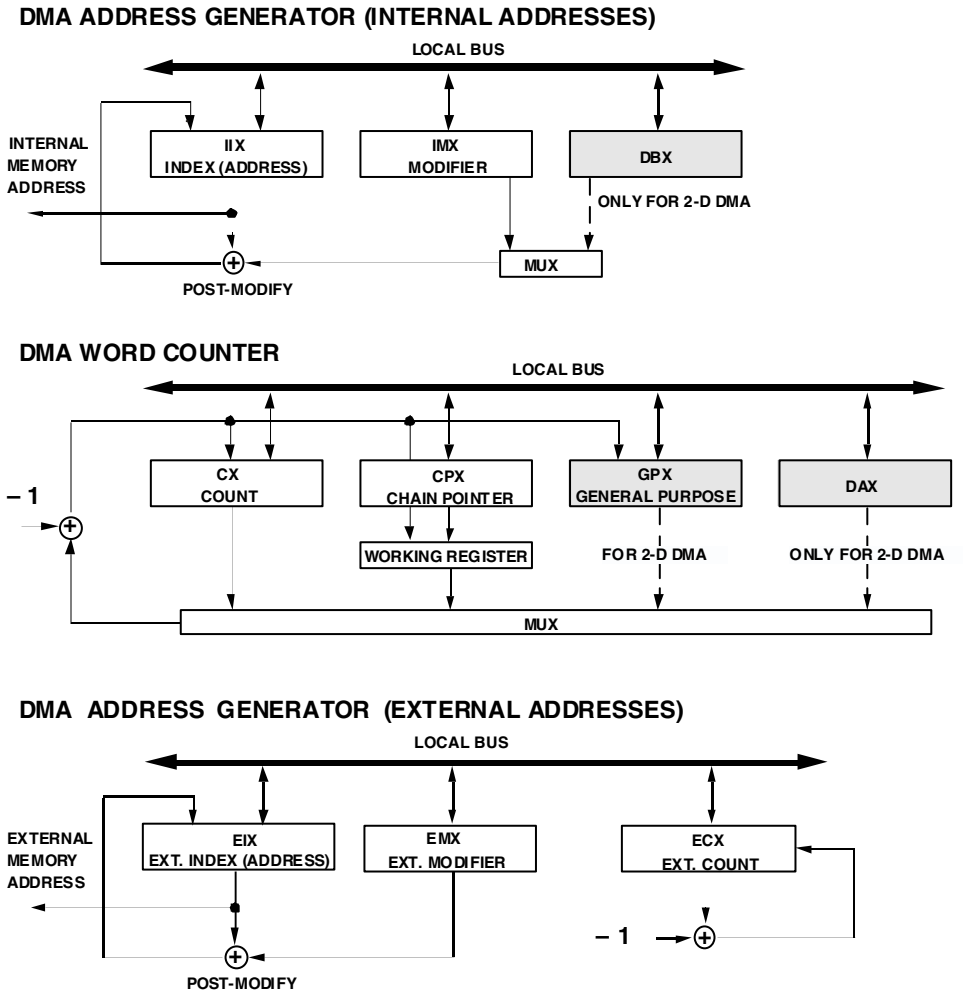


Figure 6-3. DMA Address Generator

the index register. The modify value in the IMx register is a signed integer, which allows both increment and decrement modifies.



If the I/O processor modifies the index register past the maximum 18-bit value to indicate an address out of internal memory, the index wraps around to zero. With the offset for the ADSP-21160 DSP, the wraparound address is 0x0004 0000.

Each DMA channel has a count register (Cx) that programs load with a word count to be transferred. The I/O processor decrements the count register after each DMA transfer on that channel. When the count reaches zero, the I/O processor generates the interrupt for that DMA channel. For more information on DMA interrupts, see "Using I/O Processor Status" on page 6-53.



If a program loads the count (Cx) register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before the testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit. For more information, see "External Port Channel Transfer Modes" on page 6-21, "Link Port Channel Transfer Modes" on page 6-47, or "Serial Port Channel Transfer Modes" on page 6-52.

Each DMA channel also has a chain pointer register (CPx) and a general-purpose register (GPx). Chained DMA sequences are a set of multiple DMA sequences, each autoinitializing the next in line. The location of the parameters for the next sequence comes from the CPx register. These parameters are called a Transfer Control Block, and they set up DMA parameter values for autoinitializing the next DMA sequence in the chain. Programs can use the GP register for any purpose, but usually programs store the address of the previous TCB in this register during chained DMA. For more information, see "Chaining DMA Processes" on page 6-70.

The external port DMA channels each contain three additional parameter registers, the external index register (EIx), external modify register (EMx), and external count register (ECx). These three registers are not available for the serial port and link port DMA channels. The I/O processor generates 32-bit external memory addresses using the EI, EM, and EC registers, during DMA transfers between internal memory and external memory or devices.



Programs must load the EC register with the count of external bus transfers in the DMA. If the external port is using word packing, the EC count differs from the number of words transferred in the DMA.

Instead of the EI, EM, and EC register, the serial port and link port DMA channels have the Dimension-A (DA) and Dimension-B (DB) registers. The I/O processor uses these registers for dimension indices during two-dimensional DMA operations. In one-dimensional DMA operations, programs may also use DA and DB as general-purpose registers. For more information, see "Using Two-Dimensional Link Port DMA" on page 6-83 or "Using Two-Dimensional Serial Port DMA" on page 6-93.

Memory mapped devices can communicate with the I/O processor using an internal DMA request/grant handshake on an external port DMA channel. Each channel has a single request and a single grant.

When a particular I/O port needs to perform transfers to or from internal memory, the channel asserts a request. The I/O processor prioritizes this request with all other valid DMA requests. The default channel priority is DMA channel 0 as highest and DMA channel 13 as lowest. Table 6-1 on page 6-11 lists the DMA channels in priority order. For more information, see "Managing DMA Channel Priority" on page 6-66.

When a channel becomes the highest priority requester, the I/O processor services the channel's request. In the next clock cycle, the I/O processor starts the DMA transfer.

If a DMA channel is disabled, the I/O processor does not service requests for that channel, whether or not the channel has data to transfer.

The DSP's 14 DMA channels are numbered as shown in Table 6-1. This table also shows the control, parameter, priority (DMA channel zero is highest and channel 13 lowest) and data buffer registers that correspond to each channel.

Table 6-1. DMA Channel Registers: Controls, Parameters and Buffers

DMA Chan#	Control Registers	Parameter Registers	Buffer Register	Description	Channel Priority
0	SRCTL0	II0, IM0, C0, CP0, GP0, DB0, DA0	RX0	Serial Port 0 Receive	Highest Priority
1	SRCTL1	II1, IM1, C1, CP1, GP1, DB1, DA1	RX1	Serial Port 1 Receive	
2	STCTL0	II2, IM2, C2, CP2, GP2, DB2, DA2	TX0	Serial Port 0 Transmit	
3	STCTL1	II3, IM3, C3, CP3, GP3, DB3, DA3	TX1	Serial Port 1 Transmit	
-		TCB Chain Loading Requests[1]			
-		External Accesses of Internal Memory (Direct Reads, Direct Writes) and IOP Registers[2]			

Table 6-1. DMA Channel Registers: Controls, Parameters and Buffers (Cont'd)

DMA Chan#	Control Registers	Parameter Registers	Buffer Register	Description	Channel Priority
4	LCTL0, LAR, LCOM	II4, IM4, C4, CP4, GP4, DB4, DA4	LBUF0	Link Buffer 0	
5		II5, IM5, C5, CP5, GP5, DB5, DA5	LBUF1	Link Buffer 1	
6		II6, IM6, C6, CP6, GP6, DB6, DA6	LBUF2	Link Buffer 2	
7	LCTL1, LAR, LCOM	II7, IM7, C7, CP7, GP7, DB7, DA7	LBUF3	Link Buffer 3	
8		II8, IM8, C8, CP8, GP8, DB8, DA8	LBUF4	Link Buffer 4	
9		II9, IM9, C9, CP9, GP9, DB9, DA9	LBUF5	Link Buffer 5	
10	DMAC10	II10, IM10, C10, CP10, GP10, EI10, EM10, EC10	EPB0	Ext. Port FIFO Buffer 0	
11[3]	DMAC11	II11, IM11, C11, CP11, GP11, EI11, EM11, EC11	EPB1	Ext. Port FIFO Buffer 1	
12[4]	DMAC12	II12, IM12, C12, CP12, GP12, EI12, EM12, EC12	EPB2	Ext. Port FIFO Buffer 2	
13	DMAC13	II13, IM13, C13, CP13, GP13, EI13, EM13, EC13	EPB3	Ext. Port FIFO Buffer 3	Lowest Priority

1 TCB chain loading is not associated with a specific DMA channel. TCB chain loading uses the I/O bus and requires prioritization.
2 Direct reads and writes are not associated with a specific DMA channel. Direct reads and writes use the I/O bus and require prioritization.
3 The $\overline{\texttt{DMAR1}}$ and $\overline{\texttt{DMAG1}}$ pins are handshake controls for DMA channel 11.
4 The $\overline{\texttt{DMAR2}}$ and $\overline{\texttt{DMAG2}}$ pins are handshake controls for DMA channel 12.

All of the I/O processor's registers are memory-mapped in the DSP's internal memory, ranging from address 0x0000 0000 to 0x0000 00FF. For more information on these registers, see "I/O Processor Registers" on page A-33.

Because the I/O processor registers are memory-mapped, the DSP and external processors (host or multiprocessor DSPs) have access to program DMA operations. A processor sets up a DMA channel by writing the transfer's parameters to the DMA parameter registers. After the IIx, IMx, and Cx registers (among others) are loaded with a starting source or destination address, an address modifier, and a word count, the processor is ready to start the DMA.

The external ports, link ports, and serial ports each have a DMA enable bit (DEN, LxDEN, or SDEN) in their channel control register. Setting this bit for a DMA channel with configured DMA parameters starts the DMA on that channel. If the parameters configure the channel to receive, the I/O processor transfers data words received at the buffer to the destination in internal memory. If the parameters configure the channel to transmit, the I/O processor transfers a word automatically from the source memory to the channel's buffer register. These transfers continue until the I/O processor transfers the selected number of words (count parameter).



To start a new (non-chained) DMA sequence after the current one is finished, programs must disable the channel (clear its DEN bit); write new parameters to the II, IM, and C registers; then enable the channel (set its DEN bit). For chained DMA operations, this disable-enable process is not necessary. For more information, see "Chaining DMA Processes" on page 6-70.

Setting I/O Processor—EPort Modes

The SYSCON, WAIT, and DMACx registers control the external port operating mode for the I/O processor. Table A-17 on page A-45 lists all the bits in SYSCON, Table A-19 on page A-49 lists all the bits in WAIT, and Table A-21 on page A-55 lists all the bits in DMACx.

The following bits control external port I/O processor modes. Except for the FLSH bit, the control bits in the DMACx registers have a one cycle effect latency (take effect on the second cycle after change). The FLSH bit has a

two cycle effect latency. Programs should not modify an active DMA channel's DMACx register; other than to disable the channel by clearing the DEN bit. For information on verifying a channel's status with the DMASTAT register, see "Using I/O Processor Status" on page 6-53.

Some other bits in SYSCON, WAIT, and DMACx setup non-DMA external port features. For information on these features, see "Setting External Port Modes" on page 7-2.

- **Boot Select Override.** SYSCON Bit 1 (BSO) This bit enables (if set, =1) or disables (if cleared, =0) access to Boot Memory Space. When BSO is set, the DSP uses the $\overline{\text{BMS}}$ select line (instead of $\overline{\text{MS3-0}}$) to perform DMA channel 10 accesses of external memory. When BSO is set, BMS will be asserted low for an external port DMA transfer and all memory selects ($\overline{\text{MSx}}$) will be disabled for DMA transfers. However, the memory selects are available (not disabled) to the DSP core for external memory accesses.

- **Host Packing Mode.** SYSCON Bits 6-5 (HPM) These bits select the external bus packing mode for host accesses:
000=no packing, 001=16-to-32/64, 010=16-to-48 (reset value), 011=32-to-48, 100=32-to-32/64.

- **Host Most Significant Word First Packing Select.** SYSCON Bit 7 (HMSWF) This bit selects the word packing order for host accesses as most-significant-word first (if set, =1) or least-significant-word first (if cleared, =0).

- **Buffer Hang Disable.** SYSCON Bit 16 (BHD) This bit controls whether the processor core proceeds (hang disabled if set, =1) or is held-off (hang enabled if cleared, =0) when the core tries to read from an empty EPBx, TXx, or LBUFx buffer or tries to write to a full EPBx, RXx, or LBUFx buffer.

- **External Port DMA Channel Priority Rotation Enable.** SYSCON Bit 19 (DCPR) This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation among external port DMA channels (channel 10-13).
- **Handshake and Idle for DMA Enable.** WAIT Bit 30 (HIDMA) This bit enables (if set, =1) or disables (if cleared, =0) adding an idle cycle after every memory access for DMAs with handshaking ($\overline{\text{DMARx}}$-$\overline{\text{DMAGx}}$).
- **External Port DMA Enable.** DMACx Bit 0 (DEN) This bit enables (if set, =1) or disables (if cleared, =0) DMA for the corresponding external port FIFO buffer (EPBx).
- **External Port DMA Chaining Enable.** DMACx Bit 1 (CHEN) This bit enables (if set, =1) or disables (if cleared, =0) DMA chaining for the corresponding external port FIFO buffer (EPBx).
- **External Port Transmit/Receive Select.** DMACx Bit 2 (TRAN) This bit selects the transfer direction (transmit if set, =1) (receive if cleared, =0) for the corresponding external port FIFO buffer (EPBx).
- **External Port Data Type Select.** DMACx Bit 5 (DTYPE) This bit selects the transfer data type (40/48=bit, 3-column if set, =1) (32/64-bit, 4-column if cleared, =0) for the corresponding external port FIFO buffer (EPBx).
- **External Port Packing Mode.** DMACx Bits 8-6 (PMODE) These bits select the packing mode for the corresponding external port FIFO buffer (EPBx) as follows: 000=No pack, 001=16 external to 32/64 internal packing, 010=16 external to 48 internal packing, 011=32 external to 48 internal packing, 100=32 external to 32/64 internal packing, 101=110=111=reserved.

- **Most Significant 16-bit Word First during packing.** DMACx Bit 9 (MSWF) When the buffer's PMODE is 001 or 010, this bit selects the packing order of 16-bit words (most significant first set, =1) (least significant first cleared, =0) for the corresponding external port FIFO buffer (EPBx).
- **Master Mode Enable.** DMACx Bit 10 (MASTER) This bit enables (if set, =1) or disables (if cleared, =0) master mode for the corresponding external port FIFO buffer (EPBx).
- **Handshake Mode Enable.** DMACx Bit 11 (HSHAKE) This bit enables (if set, =1) or disables (if cleared, =0) handshake mode for the corresponding external port FIFO buffer (EPBx).
- **External Handshake Mode Enable.** DMACx Bit 13 (EXTERN) This bit enables (if set, =1) or disables (if cleared, =0) external handshake mode for the corresponding external port FIFO buffer (EPBx).
- **External Port Bus Priority.** DMACx Bit 15 (PRIO) This bit selects the external bus access priority level (high if set, =1) (low if cleared, =0) for the corresponding external port FIFO buffer (EPBx).

Boot Memory DMA Mode

The BSO bit in the SYSCON register enables Boot Memory Select Override—a mode in which the I/O processor supports DMA access to boot memory space. When BSO is set, the DSP uses the $\overline{\text{BMS}}$ select line (instead of $\overline{\text{MS3-0}}$) to perform DMA channel 10 accesses of external memory.

When reading from 8-bit boot memory space, the DSP uses 8-to-48-bit packing. Programs most often use this feature to finish loading programs and data after the DSP completes its automatic 256-instruction boot-load.

When writing to 8-bit boot memory space, programs must use the shifter to place ordered bytes for the transfer in bits 39-32 of each long word to be written. Programs must use the shifter because there is no 8-to-48-bit

packing mode for external port writes. Programs most often use this feature to update writable boot memory data (flash memory or EEPROM).

External Port Buffer Modes

The HPM, HMSWF, PMODE, MSWF, and BHD bits in the SYSCON and DMACx registers select a buffer's packing mode and disable buffer not-ready processor core stalls. The packing mode bits (PMODE for DSP and HPM for host) select the external bus width and word size for transfers. Table 6-2 shows the available settings. Packed data or instructions are arranged in external memory according to the memory address that stems from their word size. For more information, see "Memory Organization and Word Size" on page 5-22. When data or instructions in external memory are *not packed*, the words are arranged in memory according to the external bus' data alignment. This data alignment appears in Figure 7-1 on page 7-2.

Table 6-2. DSP (PMODE) and Host (HPM) Packing Modes

PMODE or HPM	Packing Mode
000	No word packing (64-bit bus, 64-bit words)
001	16-to32/64-bit packing (16-bit bus, 32- or 64-bit words)
010	16-to-40/48-bit packing (16-bit bus, 40- or 48-bit words)
011	32-to-40/48-bit packing (32-bit bus, 40- or 48-bit words)
100	32-to-32/64-bit packing (32-bit bus, 32- or 64-bit words)

When packing is enabled, the DSP only uses the $\overline{\text{RDH}}$ and $\overline{\text{WRH}}$ strobes for accessing external memory, regardless of the least-significant-bit of the address.

The DSP (PMODE) and host (HPM) packing modes must match for correct word-packing operations in host systems.

When the packing mode (PMODE or HPM) is set for a 16-bit bus, programs should set up the 16-bit word order. The 16-bit word order bits (MSWF for

DSP and HMSWF for host) control the order of 16-bit words being packed or unpacked in the 32-, 48-, or 64-bit word being transferred. If the MSWF or HMSWF bit is set, the packing and unpacking is Most significant 16-bit word first.

In addition to selecting the packing mode for external port DSP transfers, programs must indicate the type of data in the transfer, using the Data Type (DTYPE) bit. For more information, see "External Port Channel Transfer Modes" on page 6-21.

The Buffer Hang Disable (BHD) bit lets the processor core proceed if the core tries to read from an empty EPBx, TXx, or LBUFx buffer or tries to write to a full EPBx, RXx, or LBUFx buffer. The processor core still performs buffer accesses when buffer hang is disabled (BHD=1). If the processor core attempts to read from an empty receive buffer, the core gets a repeat of the last value that was in the buffer. If the processor core attempts to write to a full buffer, the core overwrites the last value that was written to the buffer. Because these buffers are not initialized at reset, a read from a buffer that hasn't been filled since the reset returns an undefined value.



If an external port buffer's INTIO bit is set and DMA for that channel is not enabled, the external port channel is in single-word, interrupt-driven transfer mode. For more information, see "Using I/O Processor Status" on page 6-53.

External Port Channel Priority Modes

The DCPR and PRIO bits in the SYSCON and DMACx registers influence priority levels for an external port buffer and the external port in relation to external port DMA channels and external bus arbitration. For more information on prioritization operations, see "Managing DMA Channel Priority" on page 6-66.

Priority for DMA requests from external port channels can be fixed or rotated. When the DMA Channel Priority Rotate (DCPR) bit is cleared, the

lowest number external port channel has the highest priority, ranging from highest-priority channel 10 to lowest-priority channel 13.

When the DCPR bit is set, the priority levels rotate. High priority shifts to a new channel after each single-word transfer. The I/O processor services a single-word transfer then rotates priority to the next higher numbered channel. Rotation continues until the I/O processor services all four external port channels. Figure 6-4 illustrates this process as described in the following steps:

1. At reset, external port channels have priority order—from high to low—10, 11, 12, and 13.
2. The external port performs a single transfer on channel 11.
3. The I/O processor rotates channel priority, changing it to 12, 13, 10, and 11 (because rotating priority is enabled for this example, DCPR=1).

Even though the external port channel DMA priority can rotate, the interrupt priorities of all DMA channels are fixed.

When external port DMA channel priority is fixed (DCPR=0), channel 10 has the highest priority, and channel 13 has the lowest priority. But, programs can redefine this priority order by assigning one of the other channels the highest priority. To change the fixed priority sequence of the external port DMA channels, a program could use the following procedure:

1. Disable all external port DMA channels except the one which is to have lowest priority.
2. Select rotating priority.
3. Cause at least one transfer to occur on the enabled channel.
4. Disable rotating priority and re-enable all of the external port DMA channels

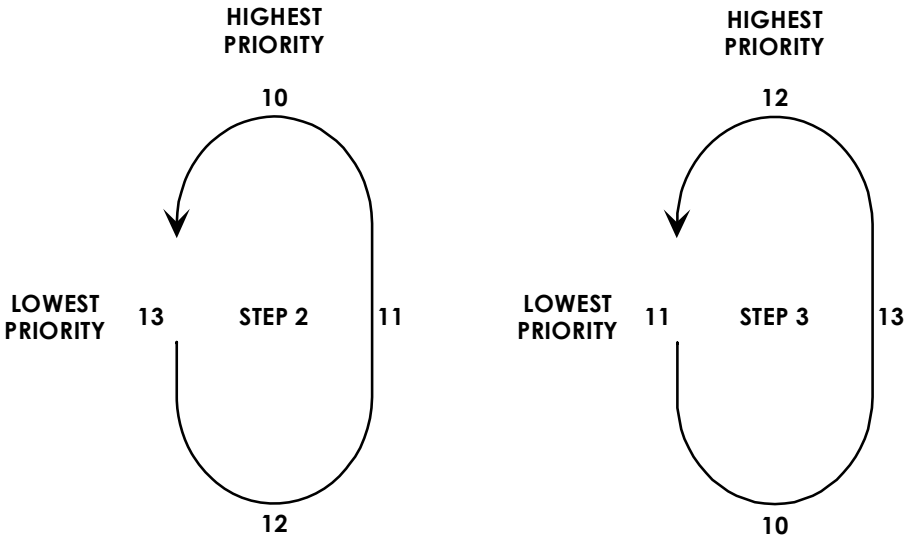


Figure 6-4. Rotating External Port DMA Channel Priority

After completing this procedure, the channel immediately after the selected channel has the highest fixed priority.

In systems where multiple processors are using the external bus, the PRIO bit raises the priority level for external port DMA transfers. When a channel's PRIO bit is set, the I/O processor asserts the Priority Access ($\overline{\text{PA}}$) pin when that channel uses the external bus. The channel gets higher priority in bus arbitration, allowing the DMA to complete more quickly.

Programs can also rotate priority between external port and link port DMA channels. For more information, see "Link Port Channel Priority Modes" on page 6-45.

External Port Channel Transfer Modes

The DEN, CHEN, TRAN, and DTYPE bits in the DMACx register enable DMA and chained DMA and select the transfer direction and data type. The DMA enable (DEN) and Chained DMA enable (CHEN) bits work together to select an external port DMA channel's transfer mode. Table 6-3 lists the modes.

Table 6-3. External Port DMA Enable Modes

CHEN	DEN	DMA Enable Mode Description
0	0	Channel disabled (chaining disabled, DMA disabled)
0	1	Single DMA mode (chaining disabled, DMA enabled)
1	0	Chain insertion mode (chaining enabled, DMA enabled, auto-chaining disabled); For more information, see "Chaining DMA Processes" on page -70.
1	1	Chained DMA mode (chaining enabled, DMA enabled, auto-chaining enabled)

Because the external port is bi-directional, the I/O processor uses the Transmit select (TRAN) bit to determine the transfer direction (transmit or receive). Data flows from internal to external memory when in transmit mode. In transmit mode, the I/O processor fills the channel's EPBx buffer when the channel's DEN bit is set.

The Data Type (DTYPE) bit determines how the DMA channel accesses columns of internal memory. If DTYPE is set, the data is 40- or 48-bit words, and the I/O processor makes 3-column internal memory accesses. If DTYPE is cleared, the data is 32- or 64-bit words, and the I/O processor makes 4-column internal memory accesses. For more information, see "Memory Organization and Word Size" on page 5-22.



The DTYPE for the transfer overrides the Internal Memory Data Width (IMDWx) setting for the internal memory block.

External Port Channel Handshake Modes

The MASTER, HSHAKE, EXTERN, and HIDMA bits in the DMACx and WAIT registers select the channel's DMA handshake and enable the hold cycles for host DMA. Table 6-4 shows how the MASTER, HSHAKE, and EXTERN bits work to select the channel's DMA handshake mode.

Table 6-4. External Port DMA Handshake Modes—DMACx MASTER (M), HSHAKE (H), and EXTERN (E) Bits

EHM	DMA Mode of Operation
000	**Slave Mode.** The DSP responds to the buffer's internal memory transfer activity based on the buffer status in the FS field, generating a DMA request whenever the buffer is not empty (on receive) or is not full (on transmit). During transmit (TRAN=1), the DSP fills the EPBx buffer when the program enables the buffer (DEN=1). For more information, see "Slave Mode" on page 6-30.
001	**Master Mode.** The DSP attempts the internal memory DMA transfers indicated by the DMA counter (Cx) based on the buffer status in the FS field, making transfers whenever the buffer is not empty (on receive) or is not full (on transmit). Systems using Master Mode should de-assert corresponding DMA request inputs, de-asserting $\overline{\text{DMAR1}}$ if channel 11 is in master mode and de-asserting $\overline{\text{DMAR2}}$ if channel 12 is in master mode. For more information, see "Master Mode" on page 6-24.
010	**Handshake Mode**. When in this mode, the DSP generates a DMA request whenever the external device asserts the DMARx pin, then the DSP asserts the DMAGx pin, transferring the data (and de-asserting DMAGx) when the external devices de-asserts the DMARx pin. Note that this mode only applies to external port buffers EPB1 and EPB2 and only applies to DMA channels 11 and 12. For more information, see "Handshake Mode" on page 6-33.
011	**Paced Master Mode**. The DSP attempts the internal memory DMA transfers indicated by the DMA counter (Cx), making transfers based on external DMA request inputs. The DSP generates a DMA request whenever the external device asserts the DMARx pin and controls the data transfer using the $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ and ACK pins and applying the selected number of waitstates. Note that this mode only applies to DMA channels 11 and 12. For more information, see "Paced Master Mode" on page 6-29.
100	Reserved

Table 6-4. External Port DMA Handshake Modes—DMACx MASTER (M), HSHAKE (H), and EXTERN (E) Bits (Cont'd)

EHM	DMA Mode of Operation
101	Reserved
110	**External-Handshake Mode.** The DSP responds to external memory DMA requests based on external DMA request inputs. This mode is identical to Handshake Mode, but applies to transfers between external memory and external devices. When in this mode, the DSP generates a DMA request whenever the external device asserts the DMARx pin, then the DSP asserts the DMAGx pin, transferring the data (and de-asserting DMAGx) when the external devices de-asserts the DMARx pin. Note that this mode only applies to external port buffers EPB1 and EPB2 and only applies to DMA channels 11 and 12. For more information, see "External-Handshake Mode" on page 6-40.
111	Reserved

For the Handshake and External-handshake modes shown in Table 6-4, programs can insert an added idle cycle after every memory access. The Handshake and Idle for DMA (HIDMA) bit in the WAIT register enables this added cycle, which reduces bus contention from devices with slow three-state timing or long recovery times.

Because external port DMA transfers can go between DSP internal memory and external memory, the I/O processor must generate addresses for both memory spaces. The external port DMA channels have additional parameter registers (EIx, EMx, ECx) for external memory access.

To support data packing options for external memory DMA transfers, the EI and EM registers can generate addresses at a different rate than the internal address registers (II and IM). This separation is shown in Figure 6-8 on page 6-68, which shows that the I/O processor has separate address generators for internal and external addresses. For this reason when packing is used for external memory DMA, the external count (EC) register indicates the number of external port transfers, not necessarily the number of internal memory words being transferred.

The DMA mode and other factors determine the size of the DMA data transfer on the external port. These other factors include the EI, EM, and EC parameters, the PMODE, DTYPE, and MAXBL values in DMACx, and the transfer capacity available in the EPBx data buffer employed in the transfer. The internal I/O processor bus transfer size varies with the II, IM, and C parameters, and the PMODE, DMA mode, DTYPE, and INT32 values in DMACx.

The following sections describe these DMA modes and transfer sizes in more detail:

- "Master Mode" on page 6-24
- "Paced Master Mode" on page 6-29
- "Slave Mode" on page 6-30
- "Handshake Mode" on page 6-33
- "External-Handshake Mode" on page 6-40

Master Mode

When the MASTER bit is set (=1) and the EXTERN and HSHAKE bits are cleared (=0) in the channel's DMACx register, the DMA channel is in master mode. A channel in this mode can independently initiate internal or external memory transfers.



Master mode applies to all external port DMA channels: 10, 11, 12, and 13.

To initiate a master mode DMA transfer, the DSP sets up the channel's parameter registers and sets the channel's DMA enable (DEN) bit. A master mode DMA channel performing internal memory to external memory data transfer automatically performs enough transfers from internal memory to keep the EPBx buffer full. When the data transfer direction is

external to internal, a master mode DMA channel also performs enough transfers from external memory to keep the EPBx buffer full.

The I/O processor uses the EI, EM, and EC registers to access external DSP memory in master mode DMA.

External Transfer Controls In Master Mode. In master mode, the DSP determines the size of the external transfer from the channel's PMODE bits and EIx, EMx, and ECx registers. Table 6-1 on page 6-11 shows the packing mode selected by the PMODE bits, and Table 6-5 on page 6-25 shows the external transfer size in master mode that results from the combination of the PMODE bits.

Table 6-5. Master Mode External Transfer Size

Transfer Size	64-bit[1]	64-bit[2]	32-bit	16-bit
PMODE	000	000	000[3], 011, 100	001, 010
EI	64-bit aligned[4]	64-bit aligned	X[5]	X
EM	0 or 1	2	X	X
EC	even # of 32-bit words, >= 2	# of 48-bit words	X	# of 16-bit xfers
DTYPE	0	1	X	X
EPBx Depth	>1	>1	>=1	>=1

1 Including packed instructions
2 Including unpacked instructions or 40-bit data
3 For PMODE=000, even 32-bit addresses (EI[0]=0) access the lower 32-bits of the data bus.
4 For a 64-bit aligned address, EI[0]=0.
5 An X indicates any supported value.

64-bit External Transfers. To enable 64-bit transfers, PMODE must be set to 000. EI must be a 64-bit aligned Normal word address, because unaligned 64-bit external transfers are not supported. EM is restricted to values of 0 (to address a memory-mapped data FIFO such as the EPBx data buffer of another DSP), or 1 (to increment through contiguous memory). EC con-

tains the number of 32-bit words to transfer. For 64-bit transfers (only), EC should be programmed to an even number. If EC must be set to an odd value, the last transfer will be a 32-bit only transfer. There must be at least two 32-bit EPBx FIFO entries available to support the 64-bit external transfer.

64-bit External Burst Transfers. Burst transfers are a subset of 64-bit transfers. In addition to the 64-bit transfer requirements described above, bursting must be enabled by setting the MAXBL field in the DMAC. Also, the burst truncates (or does not start), if the least significant bits of the 64-bit address (ADDR bits 2-1) are both set (EI bits 2-1=11) (see the SBSRAM discussion in the *External Memory* chapter for more discussion on burst address boundaries.)

Note that the external memory addressed by the burst transfer must map to a memory bank configured for synchronous access mode (with the WAIT register). The DMA programmer must ensure that the burst transfer does not straddle, or cross, the external memory bank boundaries.

The following information applies to 64-bit burst transfers from Table 6-5.

MAXBL=01

where:

- EI must address a memory bank configured for synchronous access modes (with WAIT register), and EIx must be 64-bit aligned (EIx bits 2-1 may not be = 11).
- Burst writes only supported in 1-wait write access mode. Bursts are truncated at modulo4 boundaries of the 64-bit address EPBx Depth >3 to support burst transfers

64-bit External Transfers of 48-bit Data/Instructions. Because the DSP's external bus does not support a 48-bit transfer size, programs must use 64-bit transfers sizes to move instructions. The two 64-bit transfer col-

umns in Table 6-5 describe how to transfer packed instructions or unpacked instructions.

Instructions are “packed” in external memory when the 3-column instructions are stored using all four 16-bit memory columns (same arrangement as in internal memory). When instructions are packed in external memory, the DSP cannot fetch and execute these instructions. The advantages of using packed instructions are that they take up 1/3 less memory space than unpacked and the DSP can performs 1/3 fewer bus transfers to DMA a block of these instructions than unpacked.

Instructions are “unpacked” in external memory when the 3-column instructions are stored left-aligned, using three of the 16-bit memory columns. This arrangement matches the external port bus alignment shown in Figure 7-1 on page 7-2. When instructions are unpacked in external memory, the DSP can fetch and execute these instructions. For more information, see “Executing Instructions From External Memory” on page 7-49.

32-bit External Transfers. The DSP performs 32-bit transfers when PMODE=000 (No hardware packing mode), 011 (32-to-48-bit internal), or 100 (32-bit external-to-32-bit/64-bit internal). In PMODE=000 mode, the external bus operation transfers 32-bits, instead of 64-bits if EIx, ECx, or EMx do not match the 64-bit transfer conditions in Table 6-5.

For 32-bit transfers in the PMODE=000 case, consecutive 32-bit transfers access alternating high and low halves of the 64-bit data bus.

In PMODE=011or 100, all data transfers across the upper word of the data bus (DATA63-32) as indicated in Figure 7-1 on page 7-2. This mode supports all values of EI, EM, and EC. EC contains the number of 32-bit words to transfer. There must be at least one 32-bit EPBx FIFO entry available to support the 32-bit external transfer.

16-bit External Transfers. The DSP performs 16-bit transfers when PMODE=001 (16-bit external-to-32/64-bit internal) or 010 (16-bit external-to-48-bit internal). This mode supports all values of EI, EM, and EC. EC

is programmed to the number of 16-bit words to transfer. There must be at least one 32-bit EPBx FIFO entry available to support the 16-bit external transfer. In PMODE=001, or 010, all data transfers across DATA47-32 as indicated in Figure 7-1 on page 7-2.

Internal Address/Transfer Size Generation. In master mode, the DSP determines the size of the internal transfer from the channel's PMODE bits and IIx, IMx, and Cx registers. Table 6-2 on page 6-17 shows the packing mode selected by the PMODE bits, and Table 6-6 shows the internal transfer size in master mode that results from the combination of the PMODE bits.

Table 6-6. Master Mode Internal Transfer Size Determination

Transfer Size	64-bit[1]	48-bit	32-bit
PMODE	000, 001, 100	000, 010, 011	000, 001, 100
II	depends on IM[2]	X[3]	X
IM	-1 or 1	X	X
C	even # of 32-bit words	# of 48-bit words	X
DTYPE	0	0 or 1[4]	0
EPBx Depth	>1	>1	>=1
INT32	0	0	0 or 1

1 Including packed instructions.
2 If IMx is 1 for increment, IIx must be an even, 64-bit aligned Normal word address.
If IMx is -1 for decrement, IIx must be an odd, Normal word address.
3 X indicates any supported value.
4 DTYPE=1 for PMODE=000, 48-bit instruction transfers (unpacked).
DTYPE=0 for 48-bit packing modes.

64-bit Internal Transfers. To enable internal 64-bit transfers and increment the internal IIx pointer, programs must set IIx to match the IMx selection as shown in Table 6-6. Cx contains the number of 32-bit words to transfer, and should be set to an even # of 32-bit words. The DSP decrements Cx by 2 for each 64-bit transfer. For 64-bit transfers, PMODE must be set to 000, 001 (16-bit-to-32/64-bit internal), or 100 (32-bit exter-

nal-to-32/64-bit internal). DTYPE and INT32 must be cleared. There must be at least two 32-bit EPBx FIFO entries available to support the 64-bit external transfer.

48-bit Internal Transfers. The DSP can perform 48-bit internal transfers for DMA of packed or unpacked 48-bit instructions. For more information on packed and unpacked instructions, see the discussion on page 6-26.

Many applications can use internal 64-bit transfer for 48-bit instructions. This technique can provide greater throughput than 48-bit internal transfers, but there are some restrictions. For more information on internal 64-bit transfers, see Table 6-6 and the discussion on page 6-28.

In either of the 48-bit internal transfer modes in Table 6-6 (PMODE=000 and DTYPE=1 or PMODE=010 or 011 and DTYPE=0), the DSP accesses the memory using instruction alignment (3-column read or write) for the EPBx buffer. In this case, IIx points to 48-bit words, and Cx counts the number of 48-bit internal transfers.

32-bit Internal Transfers. The DSP performs according to the conditions in Table 6-6. Under these additional conditions, the DSP performs 32-bit transfers instead of 64- or 48-bit transfers: PMODE=000 (no hardware packing), 001 (16-bit external-to-32-bit internal), or 100 (32-bit external-to-32-bit internal), and II is not aligned to a 64-bit boundary, or IM is < -1, or > 1, or C is < 2, or EPBx depth < 2, or INT32 = 1, and DTYPE=0.

Paced Master Mode

When the MASTER and HSHAKE bits are set (=1) and the EXTERN bit is cleared (=0) in the channel's DMACx register, the DMA channel is in Paced Master mode. A channel in this mode can independently initiate internal or external memory transfers.



Paced Master mode applies only to external port DMA channels 11 and 12.

In Paced Master mode, the DSP has the same control for address generation and transfer size as in master mode. For more information, see "Master Mode" on page 6-24. The difference between these modes is that in Paced Master mode external transfers are controlled and initiated (paced) by the DMARx signal as in Handshake mode. For more information, see "Handshake Mode" on page 6-33.

The DSP responds to the DMARx request only with the RDH/L, or WRH/L strobes, depending on direction and data alignment. DMAGx is not asserted in Paced Master mode. This method lets the DSP share the same buffer between the I/O processor and processor core without external gating. Paced Master mode accesses can be extended by the ACK input, by wait-states programmed in the WAIT register, and by holding the DMARx input low.

Slave Mode

When the MASTER, HSHAKE, and EXTERN bits in the channel's DMACx register are cleared (=0), the DMA channel is in slave mode. A channel in this mode cannot independently initiate external memory transfers.

To initiate a slave mode DMA transfer, an external device must read or write the channel's EPBx buffer. A slave mode DMA channel performing internal to external data transfer automatically performs enough transfers from internal memory to keep the EPBx buffer full. When the data transfer direction is external to internal, a slave mode DMA channel does not initiate any internal DMA transfers until the external device writes data to the channel's EPBx buffer.



The I/O processor does not use the EI, EM, and EC registers in slave mode DMA.

The following sequence describes a typical external to internal slave mode DMA operation where an external device transfers a block of data into the DSP's internal memory:

1. The external device writes the DMA channel's parameter registers (IIx, IMx, and Cx) and DMACx control register, initializing the channel.
2. The external device begins writing data to the EPBx buffer.
3. The EPBx buffer detects data is present and asserts an internal DMA request to the I/O processor.
4. The I/O processor grants the request and performs the internal DMA transfer, emptying the EPBx buffer FIFO.

If the internal DMA transfer is held off, the external device can continue writing to the EPBx buffer because of its eight-deep FIFO. When the EPBx FIFO becomes full, the DSP holds off the external device with the ACK signal (for synchronous accesses) or with the REDY signal (for asynchronous, host-driven accesses). This hold-off state continues until the I/O processor finishes the internal DMA transfer, freeing space in the EPBx buffer.

The following sequence describes a typical internal to external slave mode DMA operation where an external device transfers a block of data from the DSP's internal memory:

1. The external device writes the DMA channel's parameter registers (IIx, IMx, and Cx) and DMACx control register, initializing the channel and automatically asserting an internal DMA request to the I/O processor.
2. The I/O processor grants the request and performs the internal DMA transfer, filling the EPBx buffers FIFO.

3. The external device begins reading data from the EPBx buffer.
4. The EPBx buffer detects that there is room in the buffer (it is now "partially empty) and asserts another internal DMA request to the I/O processor, continuing the process.

If the internal DMA transfers cannot fill the EPBx FIFO buffer at the same rate as the external device empties it, the DSP holds off the external device with the ACK signal (for synchronous accesses) or with the REDY signal (for asynchronous, host-driven accesses) until valid data can be transferred to the EPBx buffer.



The DSP only deasserts the ACK (or REDY) signal when the EPBx FIFO buffer (or pad data buffer) is full during a write. The ACK (or REDY) signal remains asserted at the end of a completed block transfer if the EPBx buffer is not full. For reads, the DSP deasserts the ACK (or REDY) signal for each read to handle the latency of the read versus posting the write to a buffer.

In slave mode, the DSP determines the size of the transfer by decoding the read and write ($\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$) signals in addition to the channel's PMODE bits. Table 6-2 on page 6-17 shows the packing mode selected by

the PMODE bits, and Table 6-7 shows the transfer size in slave mode that results from the combination of the read and write signals and PMODE bits.

Table 6-7. Slave Mode Transfer Size Determination

Transfer Size (external↔internal)	64-bit↔ 64-bit	32-bit↔ 32-bit[1]	32-bit↔ 32/64-bit	32-bit↔ 48-bit	16-bit↔ 32/64-bit[2]	16-bit↔ 48-bit
PMODE	000	000	100	011	010	001
$\overline{\text{RDH}}$	√	√	√	Not supported	√	Not supported
$\overline{\text{RDL}}$	√	×	×		×	
$\overline{\text{WRH}}$	√	√	√		√	
$\overline{\text{WRL}}$	√	¥	×		×	
DTYPE	0	0	0	1	0	1

1 External device must be connected to the upper half of the data bus (Data[63:32])
2 External device must be connected to Data[47:32])

The DSP does not support 48-bit accesses to internal memory in slave mode.

Handshake Mode

When the MASTER and EXTERN bits are cleared (=0) and the HSHAKE bit is set (=1) in the channel's DMACx register, the DMA channel is in Handshake mode. A channel in this mode cannot independently initiate external memory transfers.

Handshake mode only applies to DMA channels 11 and 12.

To initiate a Handshake mode DMA transfer, an external device must assert an external DMA request, asserting $\overline{\text{DMAR1}}$ for access to EPB1 or $\overline{\text{DMAR2}}$ for access to EPB2. The buffers pass these request to the I/O processor, which prioritizes these requests with other internal DMA requests. When the external DMA request has the highest priority, the I/O proces-

sor asserts an external DMA grant, asserting $\overline{\text{DMAG1}}$ for EPB1 or $\overline{\text{DMAG2}}$ for EPB2. The grant signals the external device to read or write the EPBx buffer. A Handshake mode DMA channel performing internal to external data transfer automatically performs enough transfers from internal memory to keep the EPBx buffer full. When the data transfer direction is external to internal, a Handshake mode DMA channel does not initiate any internal DMA transfers until the external devices writes data to the channel's EPBx buffer.



The I/O processor does not use the EI or EM registers in Handshake mode DMA.

Other than the $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake, Handshake mode DMA operations follow almost the same process as slave mode DMA operations. The exception is that in Handshake mode DMAs from internal to external memory the external device must load the channel's ECx register with the number of external bus transfers.

In Handshake mode, the DSP determines the size of the transfer from the channel's parameter registers and PMODE bits. Table 6-2 on page 6-17 shows the packing mode selected by the PMODE bits, and Table 6-8 shows the transfer size in slave mode that results from the combination of the read and write signals and PMODE bits.

Signal timing for Handshake mode does not delay the DMA operation. The $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake operates asynchronously at up to the full CLKIN speed of the DSP. For Handshake mode DMA, the DSP does not assert the $\overline{\text{MS3-0}}$ memory select lines (address strobes). For information on $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake timing, see Figure 6-5 on page 6-36.

The I/O processor uses the rising and falling edges of $\overline{\text{DMARx}}$ in the $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake as prompts for DMA operations. On the falling edge of $\overline{\text{DMARx}}$, the edge signals the I/O processor to begin a DMA access. On the rising edge of $\overline{\text{DMARx}}$, the edge signals the I/O processor to complete the DMA access.

Table 6-8. Handshake mode Transfer Size Determination

Transfer Size (external↔internal)	64-bit↔ 64-bit	64-bit↔ 48-bit	32-bit↔ 32/64-bit[1]	32-bit↔ 48-bit[2]	16-bit↔ 32/64-bit[2]	16-bit↔ 48-bit[2]
PMODE	000	000	100	011	001	001
IIx	64-bit aligned	X	X	X	X	X
IMx	1	X	X	X	X	X
Cx	# of 32-bit words[3]	# of 48-bit words[4]	# of 32-bit words	# of 48-bit words	# of 32-bit words	# of 48-bit words
ECx	# of 32-bit words[1]	3/4 * Cx	# of 32-bit words	# of 32-bit words	# of 16-bit words	# of 16-bit words
DTYPE	0	1	0	0	0	0

1 External device must be connected to the upper half of the data bus (Data[63:32])
2 External device must be connected to Data[47:32])
3 Must be an even number of 32-bit words
4 Should be a multiple of 4

The following sequence describes the process for requesting access to an EPBx buffer in Handshake mode:

1. The external device asserts the buffer's $\overline{\text{DMARx}}$ signal, placing an external DMA request for access to the EPBx buffer.
2. The EPBx buffer detects the falling edge of the $\overline{\text{DMARx}}$ signal and passes the external DMA request to the I/O processor, synchronizing the DMA operation with the processor's system clock.
3. To be recognized in a particular cycle, the $\overline{\text{DMARx}}$ low transition must meet the signal setup time from the DSP data sheet. If the transition is slower than the setup time, the signal may not take effect until the following cycle.

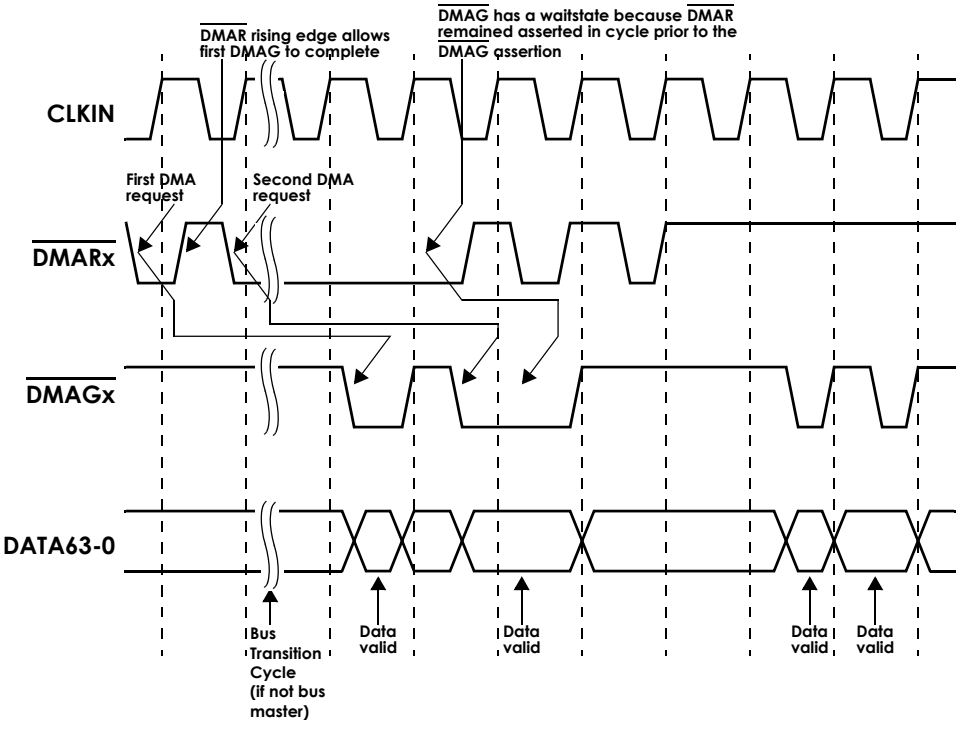


Figure 6-5. Handshake DMA Timing (Asynchronous Requests)

4. The I/O processor prioritizes the external DMA request with other internal DMA requests. If the DSP is not already bus master, the DSP arbitrates for the external bus when the external DMA request has the highest priority, unless the EPBx buffer is blocked.

5. If the EPBx buffer is full during a write or empty during a read, the buffer is blocked. The DSP does not begin external bus arbitration until the I/O processor services the EPBx buffer, returning it to the unblocked state empty for writing or full for reading.

6. The DSP becomes bus master and asserts $\overline{\text{DMAGx}}$.
7. The DSP keeps $\overline{\text{DMAGx}}$ asserted until the cycle after the external device deasserts $\overline{\text{DMARx}}$. By holding $\overline{\text{DMARx}}$ asserted, the external device holds the DSP until the external device is ready to proceed. If the external device does not need to extend the DMA grant cycle, the external device can deassert $\overline{\text{DMARx}}$ immediately (not waiting for $\overline{\text{DMAGx}}$), providing the $\overline{\text{DMARx}}$ assertion time meets the timing requirements from the DSP data sheet. The responding $\overline{\text{DMAGx}}$ in this case is a short pulse, and the DSP only uses the external bus for one cycle.

Because the I/O processor has a three-cycle DMA pipeline and a seven-deep external request counter, the DSP can execute $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake operations at up to the full CLKIN rate of the DSP. The I/O processor has a three-cycle DMA pipeline, similar to the program sequencer's fetch–decode–execute instruction pipeline.

The I/O processor processes the DMA pipeline in the following stages:

- Recognizes the DMA request and arbitrates internal DMA priority during the DMA fetch cycle.
- Generates the DMA address and arbitrates external bus access during the DMA decode cycle.
- Transfers DMA data during the DMA execute cycle.



Because the I/O processor has a three-cycle DMA pipeline, there is a minimum delay of three cycles before the DSP asserts $\overline{\text{DMAGx}}$. This delay is in addition to any delay from internal DMA arbitration, so the external device must not assume that the DMA grant can arrive within two cycles even if higher priority DMA operations are disabled and the external bus is available for the transfer.

The I/O processor's external request counter increments each time the external device asserts $\overline{\text{DMARx}}$ and decrements each time the DSP replies by

asserting $\overline{\text{DMAGx}}$. The external request counter records up to seven requests, so the external device can make up to seven requests before the first one has been serviced.

If the DSP cannot immediately service the DMA requests in the external request counter, the DSP services the requests on a prioritized basis. The external DMA device is responsible for keeping track of requests, monitoring grants, and pipelining the data when operating at full speed.



If the external device makes more than seven $\overline{\text{DMARx}}$ without receiving a grant, the delayed grant causes unpredictable results.

The DSP only asserts $\overline{\text{DMAGx}}$ for the number of $\overline{\text{DMARx}}$ requests indicated by the external request counter. If the external devices makes more requests than the count indicates, the DSP $\overline{\text{DMAGx}}$ assertions cannot match the number of external device requests. To clear this mismatch, programs can clear the buffer and the external request counter using the flush bit ($\overline{\text{FLSH}}$) in the channel's $\overline{\text{DMACx}}$ control register.

To prevent holding off the DSP, the external device must service the DSP's data requirements when the DSP's asserts the $\overline{\text{DMAGx}}$ grant signal. The external device should immediately supply data for writes to the DSP or immediately accept data on reads from the DSP. External interfaces can handle this I/O by placing the data in an external FIFO. When performing DMA operations at the full CLKIN speed of the DSP, the system may need a three-deep external FIFO to handle the latency between request and grant. Programs on the external device can optimize operation of this FIFO by issuing three requests rapidly and making the next requests conditional on when the DSP issues a grant.

The external devices must follow the conditions in Figure 6-6 on page 6-39 when enabling or disabling Handshake mode for an external port DMA channel.

The DSP ignores a disabled (transitioning from disabled to enabled) DMA channel's $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ pins and ignores internal $\overline{\text{DMARx}}$ asser-

tions for up to two processor core clock cycles after the instruction that enables the channel in Handshake mode.

- The external devices must maintain $\overline{\text{DMARx}}$ deasserted (kept high, not low or changing) during the instruction that enables DMA in handshake mode. Before using the channel for the first time, programs flush the DMA channel, asserting the FLSH bit in the DMACx control register. This action is not required during chain insertion.
- The DSP deasserts $\overline{\text{DMAGx}}$ if a program disables the channel while $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ are asserted (=0). This action clears the channel's active status bit, avoiding a potential deadlock condition.

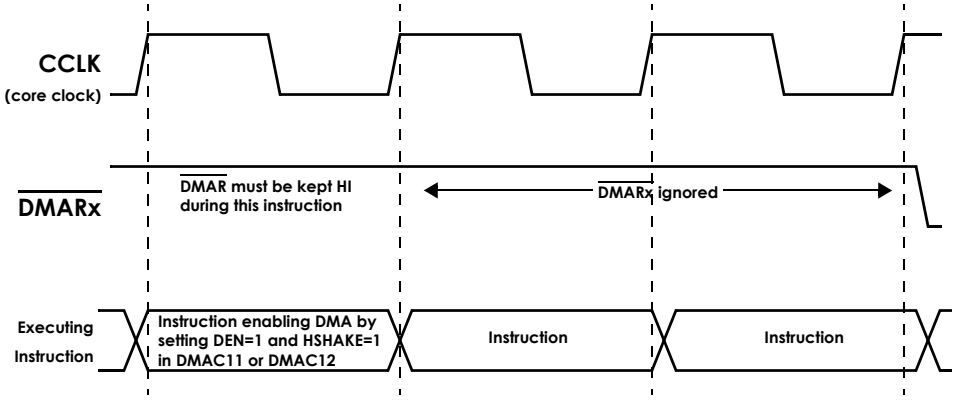


Figure 6-6. DMARx Delay After Enabling Handshake DMA

DSPs in a multiprocessing cluster may share a $\overline{\text{DMAGx}}$ signal, because only the bus master drives $\overline{\text{DMAGx}}$. On the bus slaves, $\overline{\text{DMAGx}}$ is three-stated. This state eliminates the need for external gating if more than one DSP or the host needs to drive the DMA buffer. Systems may need a pull-up resistor on this line if the host is not connected to the pin or does not drive it when it acquires the bus. $\overline{\text{DMAGx}}$ has the same timing and transitions as the $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes in asynchronous access mode. $\overline{\text{DMAGx}}$ responds to the $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ signals in the same way as the read and write strobes. For more information, see "External Port".

External-Handshake Mode

When the MASTER bit is cleared (=0) and the HSHAKE and EXTERN bits are set (=1) in the channel's DMACx register, the DMA channel is in the External-Handshake mode. A channel in this mode cannot independently initiate external memory transfers.

External-handshake mode is identical to Handshake mode, except that External-Handshake mode transfers data between external memory and an external device. This section covers the differences between Handshake mode and External-Handshake mode. For more information, see "Handshake Mode" on page 6-33.

Like Handshake mode, External-Handshake mode only applies to DMA channels 11 and 12.

To initiate an External-Handshake mode DMA transfer, an external device must assert an external DMA request, asserting $\overline{\text{DMAR1}}$ for access to DMA channel 11or $\overline{\text{DMAR2}}$ for access to DMA channel 12. The channels pass these request to the I/O processor, which prioritizes these requests with other internal DMA requests. When the external DMA request has the highest priority, the I/O processor asserts an external DMA grant, asserting $\overline{\text{DMAG1}}$ for channel 11 or $\overline{\text{DMAG2}}$ for channel 12. The grant signals the external device to read or write the external bus. An External-Handshake mode DMA channel performing external to external data transfer automatically generates external memory addresses and strobes for transfers between external memory and the external device.

Unlike Handshake mode, the I/O processor must use the EIx, EMx, and ECx registers in External-Handshake mode DMA. Also unlike Handshake mode, the data for DMA channels 11 and 12 does not pass through the EPB1 or EPB2 buffers.

During External-Handshake mode transfers, the I/O processor generates external memory access cycles. $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ operate the same as in Handshake mode, but the DSP also outputs addresses, $\overline{\text{MS3-0}}$ memory selects, and $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes, and responds to ACK. On external

memory writes, the DSP asserts $\overline{\text{DMAGx}}$ until the external device releases the ACK line or any DSP waitstates expire. The external memory access by the external devices responds as if the DSP processor core were making the access. See "External Port" for information on connecting external devices to the external port.

Because the I/O processor accesses external memory in External-Handshake mode, programs must load the DMA channel's EIx, EMx, and ECx parameter registers, and the DMACx PMODE bits. These settings let the I/O processor generate the external memory addresses and word count.

External-handshake mode does not support chained DMA interrupts. Because no internal DMA transfers occur in External-Handshake mode, the PCI bit in the channel's CPx register cannot disable the DMA interrupt. Programs must use the IMASK register to mask this interrupt.

In External-Handshake mode, the DSP does not perform packing. The DSP does determine the size of the transfer from the channel's parameter registers, PMODE bits. Table 6-9 shows the transfer size in slave mode that results from the combination of the read and write signals and PMODE bits.

Setting I/O Processor—LPort Modes

The SYSCON, LCOM, LAR, and LCTLx registers control the link ports operating modes for the I/O processor. Table A-17 on page A-45 lists all the bits in SYSCON, Table A-24 on page A-65 lists all the bits in LCOM, Table A-25 on page A-67 lists all the bits in LAR, and Table A-23 on page A-63 lists all the bits in LCTLx.

Table 6-9. External-Handshake Mode Transfer Size Determination

Transfer Size (DSP↔device)	64-bit Memory ↔ 64-bit Device	64-bit Memory ↔ 64-bit Device	32-bit Memory ↔ 32-bit Device	32-bit Memory ↔ 32-bit Device
PMODE	000	000	000	100
EIx	X	constant	X	X[1]
EMx	1[2]	0[3]	Not 0 or 1	X
ECx	Even[4]	Even4	X[5]	X5
DTYPE	0	0	0	0

- For 64-bit transfers, the device must be connected to DATA63-0, and both RDH/L and WRH/L are used.
- For 32-bit transfers, the device may be connected to either DATA63-32 or DATA31-0 and the corresponding $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ is used depending on the address.
- X indicates any legal value.

1 For packed 32-bit transfers (PMODE=100), all ADDR bits are decoded.
2 For EMx=1, EIx is incremented by 2.
3 For EMx=0, EIx remains constant.
4 For 64-bit transfers, ECx is decremented by 2.
5 For 32-bit transfers, ECx is decremented by 1.

This section contains:

- "Link Port Buffer Modes" on page 6-44
- "Link Port Channel Priority Modes" on page 6-45
- "Link Port Channel Priority Modes" on page 6-45

The following bits control link port I/O processor modes. The control bits in the LCTLx registers have a one cycle effect latency (take effect on the second cycle after change). Programs should not modify an active DMA channel's bits in the LCTLx register; other than to disable the channel by

clearing the LxDEN bit. For information on verifying a channel's status with the DMASTAT register, see "Using I/O Processor Status" on page 6-53.

- **Link Port DMA Channel Priority Rotation Enable.** SYSCON Bit 10 (LDCPR) This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation among link port DMA channels (channel 4-9).
- **Link–External Port DMA Channel Priority Rotation Enable.** SYSCON Bit 12 (PRROT) This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation between link port DMA channels (channel 4-9) and external port DMA channels (channel 10-13).
- **Link port assignment for LBUFx.** LAR Bits 2-0, 5-3, 8-6, 11-9, 14-12, and 17-15 (AxLB) Each 3-bit set assigns a link port to the corresponding link buffer (LBUFx) as shown in Table 6-10 on page 6-44.
- **Link Buffer Enable.** LCTL0 Bits 0, 10, and 20 and LCTL1 Bits 0, 10, and 20 (LxEN) This bit enables (if set, =1) or disables (if cleared, =0) the corresponding link buffer (LBUFx).
- **Link Buffer DMA Enable.** LCTL0 Bits 1, 11, and 21 and LCTL1 Bits 1, 11, and 21 (LxDEN) This bit enables (if set, =1) or disables (if cleared, =0) DMA transfers for the corresponding link buffer (LBUFx).
- **Link Buffer DMA Chaining Enable.** LCTL0 Bits 2, 12, and 22 and LCTL1 Bits 2, 12, and 22 (LxCHEN) This bit enables (if set, =1) or disables (if cleared, =0) DMA chaining for the corresponding link buffer (LBUFx).
- **Link Buffer Transfer Direction.** LCTL0 Bits 3, 13, and 23 and LCTL1 Bits 3, 13, and 23 (LxTRAN) This bit selects the transfer direction (transmit if set, =1) (receive if cleared, =0) for the corresponding link buffer (LBUFx).

- **Link Buffer Extended Word Size.** LCTL0 Bits 4, 14, and 24 and LCTL1 Bits 4, 14, and 24 (LxEXT) This bit selects the transfer extended word size (48-bit if set, =1) (32-bit if cleared, =0) for the corresponding link buffer (LBUFx). Programs must not change a buffer's LxEXT setting while the buffer is enabled.
- **Link Buffer DMA 2-Dimensional.** LCTL0 Bits 7, 17, and 27 and LCTL1 Bits 7, 17, and 27 (LxDMA2D) This bit enables (if set, =1) or disables (if cleared, =0) two-dimensional DMA transfers for the corresponding link buffer (LBUFx).

Some other bits in LCTLx setup non-DMA link port features. For information on these features, see "Setting Link Port Modes" on page -5.

Link Port Buffer Modes

The AxLB and LxEN bits in the LAR and LCTLx registers assign link ports to link buffers and enable link buffers. Table 6-10 shows how the AxLB bits in the LAR register assign link buffers to link ports.

To enable a link buffer, a program sets the buffer's LxEN bit in LCTL0 or LCTL1. To disable a link buffer, a program clears the buffer's LxEN bit in LCTL0 or LCTL1. The bit locations appear on page 6-41.

When the DSP disables the buffer (LxEN transitions from high to low), the DSP clears the corresponding LxSTAT and LxRERR bits.

Table 6-10. DSP Link Buffer-to-Link Port Assignments (AxLB)

Link Buffers		Link Ports						
Buffer#	LAR Bits	0	1	2	3	4	5	NA[1]
0	2-0	000	001	010	011	100	101	111
1	5-3	000	001	010	011	100	101	111
2	8-6	000	001	010	011	100	101	111
3	11-9	000	001	010	011	100	101	111

Table 6-10. DSP Link Buffer-to-Link Port Assignments (AxLB)

Link Buffers		Link Ports						
Buffer#	**LAR Bits**	**0**	**1**	**2**	**3**	**4**	**5**	**NA[1]**
4	14-12	000	001	010	011	100	101	111
5	17-15	000	001	010	011	100	101	111

1 NA indicates Not Assigned

Link Port Channel Priority Modes

The LDCPR and PRROT bits in the SYSCON register select priority levels for the link port buffers in relation to the priority of other link port buffers and the other I/O ports. The Link Port DMA Channel Priority Rotation Enable (LDCPR) bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation among link port DMA channels (channel 4-9). Rotating priority distributes link port DMA channel access to the IO bus. When channel priority is rotating, the DSP arbitrates IO bus access between contending link port DMA channels, forcing the channels to take turns.

When channel priority is fixed, the lower numbered link port DMA channel (4-9) always has priority over the higher numbered channel when contending for IO bus access. When LDCPR is set (rotating priority), high priority shifts to a new channel after each single-word transfer. The order of channel priority then rotates. After a single-word transfer, the I/O processor rotates priority to the next higher-numbered channel, and so on until all six are serviced.

Figure 6-7 illustrates this process, according to the following example:

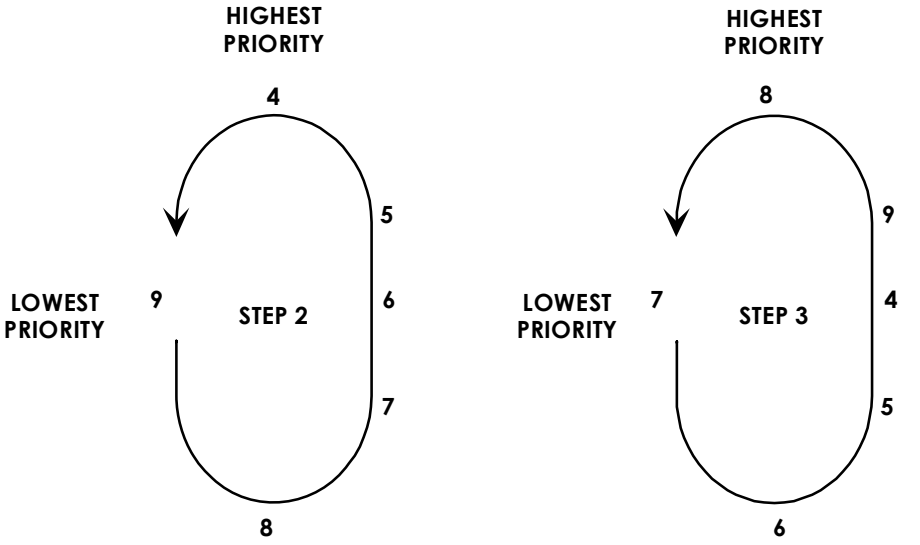


Figure 6-7. Link Port Channel Rotating Priority Example

- At reset, link port channels have priority order—from high to low—4, 5, 6, 7, 8, and 9.
- The link port performs a single transfer on channel 7.
- The I/O processor rotates channel priority, changing it to 8, 9, 4, 5, 6, and 7 (because rotating priority is enabled for this example, LDCPR=1).

Even though the link port channel DMA priority can rotate, the interrupt priorities of all DMA channels are fixed.

When a program uses fixed priority for the link port DMA channels, the I/O processor assigns the highest priority to Channel 4 and the lowest priority to Channel 9. For a list of all fixed priority assignments, see the list of DMA channels in Table 6-1 on page 6-11.

Programs can change the fixed priority order, assigning a different channel to the highest priority. The following example shows how to change the fixed priority sequence of the link port DMA channels:

- Disable all link port DMA channels except the one immediately above the channel that is to have highest priority.
- Select rotating priority (by setting the LDCPR bit).
- Cause at least one transfer to occur on the enabled channel.
- Disable rotating priority and re-enable all of the link port DMA channels.
    - The channel immediately after the selected channel now has the highest fixed priority.

Programs can also rotate priority between the link port and external port DMA channels. The Link–External Port DMA Channel Priority Rotation Enable (PRROT) bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation between link port DMA channels (channel 4-9) and external port DMA channels (channel 10-13).

Rotating priority distributes link port and external port DMA channels' access to the I/O bus. When channel priority is rotating, the DSP arbitrates IO bus access between contending link port and external port DMA channels, forcing the channel types to take turns. When channel priority is fixed, any link port DMA channel (4-9) always has priority over any external port DMA channel (10-13) when contending for IO bus access.

Link Port Channel Transfer Modes

The LxDEN, LxCHEN, LxTRAN, LxEXT, and LxDMA2D bits in the LCTLx register enable link port DMA, chained DMA, and two-dimensional DMA and select the transfer direction and format. The link DMA enable (LxDEN) and link Chained DMA enable (LxCHEN) bits work together to select a link port DMA channel's transfer mode.

Table 6-11 lists the modes.

Table 6-11. Link Port DMA Enable Modes

LxCHEN	LxDEN	DMA Enable Mode Description
0	0	Channel disabled (chaining disabled, DMA disabled)
0	1	Single DMA mode (chaining disabled, DMA enabled)
1	0	Chain insertion mode (chaining enabled, DMA enabled, auto-chaining disabled); For more information, see "Chaining DMA Processes" on page 6-70.
1	1	Chained DMA mode (chaining enabled, DMA enabled, auto-chaining enabled)

Because link ports are bi-directional, the I/O processor uses the link Transmit select (LxTRAN) bit to determine the transfer direction (transmit or receive). Data flows from internal to external memory when in transmit mode. In transmit mode, the I/O processor fills the channel's LBUFx buffer when the channel's LxDEN bit is set.

The link Extend Word Size (LxEXT) bit determines how the DMA channel accesses columns of internal memory. If LxEXT is set, the data is 40- or 48-bit words, and the I/O processor makes 3-column internal memory accesses. If LxEXT is cleared, the data is 32-bit words, and the I/O processor makes 2-column internal memory accesses. For more information, see "Memory Organization and Word Size" on page 5-22.



The LxEXT for the transfer overrides the Internal Memory Data Width (IMDWx) setting for the internal memory block.

The link buffer's DMA 2-Dimensional (LxDMA2D) bit enables (if set, =1) or disables (if cleared, =0) two-dimensional DMA transfers for the corresponding link buffer (LBUFx). When LxDMA2D is set, the channel's DAx and

DBx registers control the two-dimensional DMA process. For more information, see "Using Two-Dimensional Link Port DMA" on page 6-83.

A link buffer's LxDMA2D bit must be cleared (=0) for standard (one-dimension) DMA operation.

Setting I/O Processor—SPort Modes

The SRCTLx and STCTLx registers control the serial port operating mode for the I/O processor. Table A-28 on page A-74 lists all the bits in SRCTLx and Table A-27 on page A-71 lists all the bits in STCTLx.

This section contains:

- "Serial Port Buffer Modes" on page 6-51
- "Serial Port Channel Priority Modes" on page 6-52
- "Serial Port Channel Transfer Modes" on page 6-52

The following bits control serial port I/O processor modes. The control bits in the SRCTLx and STCTLx registers have a one cycle effect latency (take effect on the second cycle after change). Programs should not modify an active DMA channel's bits in the SRCTLx or STCTLx registers; other than to disable the channel by clearing the SDEN bit. To change an inactive serial port's operating mode, programs should clear a serial port's control register before writing new settings to the control register. For information on verifying a channel's status with the DMASTAT register, see "Using I/O Processor Status" on page 6-53.

Some other bits in SRCTLx and STCTLx setup non-DMA serial port features. For information on these features, see "Setting Serial Port Modes" on page 9-7.

- **Serial Port Enable.** SRCTLx and STCTLx Bit 0 (SPEN) This bit enables (if set, =1) or disables (if cleared, =0) the corresponding serial port.
- **Data Type Select.** SRCTLx and STCTLx Bits 2-1 (DTYPE) These bits select the data type formatting for normal and multi-channel reception as follows: (normal/multichannel= format) 00/x0=Right-justify and zero-fill unused MSBs, 01/x1=Right-justify and sign-extend unused MSBs, 10/0x=Compand using μ-law, 11/1x=Compand using A-law.
- **Serial Word Endian Select.** SRCTLx and STCTLx Bit 3 (SENDN) This bit selects little endian words (LSB first, if set, =1) or big endian words (MSB first, if cleared, =0).
- **Serial Word Length Select.** SRCTLx and STCTLx Bits 8-4 (SLEN) These bits select the word length (–1) in bits. Word sizes can be from 3-bit (SLEN=2) to 32-bit (SLEN=31).
- **16-bit to 32-bit Word Packing Enable.** SRCTLx and STCTLx Bit 9 (PACK) This bit enables (if set, =1) or disables (if cleared, =0) 16- to 32-bit word packing.
- **Serial Port Receive DMA Enable.** SRCTLx and STCTLx Bit 18 (SDEN) This bit enables (if set, =1) or disables (if cleared, =0) the serial port's receive DMA.

- **Serial Port Receive DMA Chaining Enable.** SRCTLx and STCTLx Bit 19 (SCHEN) This bit enables (if set, =1) or disables (if cleared, =0) serial port DMA chaining.
- **Two Dimensional DMA Array Enable.** SRCTLx and STCTLx Bit 21 (D2DMA) This bit enables (if set, =1) or disables (if cleared, =0) two-dimensional serial DMA. This bit must be cleared for multi-channel operation.

Serial Port Buffer Modes

The SPEN, SENDN, SLEN, and PACK bits in the SRCTLx and STCTLx registers enable the serial port and select the transfer format. To enable a serial port transmit or receive buffer, a program sets the buffer's SPEN bit in the SRCTLx or STCTLx register.

To disable a serial port transmit or receive buffer, a program clears the buffer's SPEN bit in the SRCTLx or STCTLx register.



If a serial port buffer is enabled and DMA for that channel is not enabled, the serial port is in single-word, interrupt-driven transfer mode. For more information, see "Using I/O Processor Status" on page 6-53.

Each serial port buffer allows independent settings for the three transfer format features: bit order, word length, and word packing.

For transferring little endian words (LSB first, if set, =1) to or from little endian devices, the serial port buffers have a Serial Word Endian Select (SENDN) bit. This bit selects little endian words (LSB first, if set, =1) or big endian words (MSB first, if cleared, =0).

The Serial Word Length Select (SLEN) bit field selects the transfer word length (-1) in bits. Word sizes can be from 3-bit (SLEN=2) to 32-bit (SLEN=31).

If the serial word length is 16-bits or smaller, the serial port can pack two of these words into the serial port buffer. The 16-bit to 32-bit Word Packing Enable (PACK) bit can enable this packing because the I/O processor performs 32-bit transfers between the serial port buffers and DSP memory.

In addition to selecting the endian, length, and packing modes for serial port DSP transfers, programs must indicate the type of data in the transfer, using the Data Type (DTYPE) bit. For more information, see "Serial Port Channel Transfer Modes" on page 6-52.

Serial Port Channel Priority Modes

Serial port DMA transfers always take priority over external port or link port DMA transfers. For more information on prioritization operations, see "Managing DMA Channel Priority" on page 6-66.

Serial Port Channel Transfer Modes

The SDEN, SCHEN, DTYPE, and D2DMA bits in the SRCTLx and STCTLx register enable serial port DMA, chained DMA, and two-dimensional DMA and select the format. The DMA enable (SDEN) and Chained DMA enable (SCHEN) bits work together to select a serial port DMA channel's transfer mode. Table 6-12 lists the modes.

Table 6-12. Serial Port DMA Enable Modes

SCHEN	SDEN	DMA Enable Mode Description
0	0	Channel disabled (chaining disabled, DMA disabled)
0	1	Single DMA mode (chaining disabled, DMA enabled)
1	0	Chain insertion mode (chaining enabled, DMA enabled, auto-chaining disabled); For more information, see "Chaining DMA Processes" on page 6-70.
1	1	Chained DMA mode (chaining enabled, DMA enabled, auto-chaining enabled)

Because serial port buffers are bi-directional, the I/O processor does not need an indicator to determine the transfer direction (transmit or receive). Data flows from internal to external devices using a transmit (TXx) buffer. When transmitting serial data as DMA, the I/O processor fills the channel's TXx buffer when the channel's SDEN bit is set.

The serial port buffer's DMA 2-Dimensional (D2DMA) bit enables (if set, =1) or disables (if cleared, =0) two-dimensional DMA transfers for the corresponding serial port buffer (TXx or RXx). When 2DDMA is set, the channel's DAx and DBx registers control the two-dimensional DMA process. For more information, see "Using Two-Dimensional Serial Port DMA" on page 6-93.



A serial port buffer's D2DMA bit must be cleared (=0) for standard (one-dimension) DMA operation.

Using I/O Processor Status

The I/O processor monitors the status of data transfers on DMA channels and indicates status in the DMASTAT, IRPTL, and LIRPTL registers. Table A-9 on page A-18 lists all the bits in IRPTL, Table A-10 on page A-24 lists all the bits in LIRPTL, and a discussion of DMASTAT appears in "DMA Channel Status Register (DMASTAT)" on page A-60.

The I/O processor reports on DMA in progress, DMA complete, or DMA channel not ready status:

- All DMA channels can be active or inactive. If a channel is active—a DMA is in progress on that channel—the I/O processor indicates the active status by setting the channel's bit in the DMASTAT register.
- When an unchained (single-block) DMA process reaches completion on any DMA channel, the I/O processor generates that DMA channel's interrupt—sets the DMA channel's interrupt latch bit in the IRPTL or LIRPTL register. DMA process is complete when
  - The count in Cx=0 (for slave mode and handshake modes).
  - The count in ECx=0 (for external handshake mode).
  - The count in Cx=0 and ECx=0 (for master mode and paced master mode).
- When a DMA process in a chained DMA sequence reaches completion (the count in Cx=0) on any DMA channel, the I/O processor generates an interrupt if the PCI bit in the channels CPx register is set, except in external-handshake mode. The I/O processor also generates that DMA channel's interrupt when the last block in a chained DMA reaches completion whether or not PCI is set.
- When a DMA channel's buffer not being used for a DMA process, the I/O processor can generate an interrupt on single word writes or reads of the buffer. This interrupt service differs slightly for each port. For more information on single-word interrupt-driven transfers, see "External Port Status" on page 6-57, "Link Port Status" on page 6-60, and "Serial Port Status" on page 6-63.

Using the DMA Channel Status Register (DMASTAT), programs can check which DMA channels are performing a DMA or chained DMA. For each channel, the I/O processor sets the channel's active status bit if DMA for

that channel is enabled and a DMA sequence is in progress on that channel. The I/O processor sets the channel's chaining status bit if a chained DMA sequence is in progress or pending on that channel.



There is a one cycle latency between a change in DMA channel status and the status update in the DMASTAT register.

As an alternative to interrupt-driven DMA, programs can poll the DMASTAT register to determine when a single DMA sequence is done. To poll channel status, programs read DMASTAT. If both status bits for the channel are cleared, the DMA sequence has completed.



If chaining is enabled on a DMA channel programs should not use polling to determine channel status. Polling could provide inaccurate information in this case because the next DMA sequence might be under way by the time the polled status is returned.

During interrupt-driven DMA, programs use the interrupt mask bits in the IMASK and LIRPTL registers to selectively mask DMA channel interrupts that the I/O processor latches into the IRPTL and LIRPTL registers. Descriptions of these conditions appear on on page 6-53.



The I/O processor only generates a DMA complete interrupt when the channel's count register decrements to zero as a result of actual DMA transfers. Writing zero to a count register does not generate the interrupt.

A channel interrupt mask in IMASK and IRPTL masks out DMA complete interrupts for a channel, but other types of interrupt masking are also available. These other types of interrupt masking include:

- By clearing a channels PCI bit during chained DMA, programs mask the DMA complete interrupt for a DMA processes within a chained DMA sequence.
- By masking the LPISUM interrupt, programs mask out the logical Or'ing of link port interrupt status.
- By masking the LSRQ interrupt, programs mask out link port service requests to link ports that do not have an assigned link buffer.

These lower levels of interrupt masking let programs limit some of the conditions that can cause DMA channel interrupts.



Each DMA channel has its own interrupt. Although the external port and link port channel access priority can rotate, the interrupt priorities of all DMA channels are fixed.

In DSP systems using I/O processor interrupts, an external device may need to change the DSP's interrupt mask. This task presents a challenge because the IMASK register is not memory-mapped and is not directly accessible to external devices through the external port. To read or write IMASK through the external port, programs can set up an interrupt vector routine to handle this task. The VIRPT vector interrupt register may be used for this task.

The I/O processor can also generate DMA channel interrupts for I/O port operations that do not use DMA. In this case, the I/O processor generates a DMA interrupt when data becomes available at the receive buffer or when the transmit buffer does not have new data to transmit. Generating DMA interrupts in this fashion lets programs implement interrupt-driven I/O under control of the processor core. Care is needed because multiple interrupts can occur if several I/O ports transmit or receive data in the same cycle.

For more information on these types of single-word interrupt-driven transfers, see the external port discussion on page 6-60, link port discussion on page 6-63, or serial port discussion on page 6-65.

External Port Status

The I/O processor monitors the status of data transfers on the external port. DMA channel status for the external port is described in "Using I/O Processor Status" on page 6-53. This section describes external port specific status features, such as buffer status, buffer control, and single-word interrupt-driven transfers.

Bits in the SYSTAT, SYSCON and DMACx registers indicate and control status of external port buffers. Table A-20 on page A-51 lists all the bits in SYSTAT, Table A-17 on page A-45 lists all the bits in SYSCON, and Table A-21 on page A-55 lists all the bits in DMACx. The following bits influence external port buffer status:

- **Host Packing Status.** SYSTAT bits 15-14 (HPS). These bits indicate the host's packing status.
- **Host Packing Status Flush.** SYSCON Bit 8 (HPFLSH) This bit flushes (when set, =1) settings for the direct write FIFO.
- **External Port Packing Status.** DMACx Bits 4-3 (PS). These bits indicate the corresponding FIFO buffer's packing status.
- **Single-Word Interrupt Enable.** DMACx Bit 12 (INTIO). This bit enables (if set, =1) or disables (if cleared, =0) single-word, non-DMA, interrupt-driven transfers for the corresponding external port FIFO buffer (EPBx). To avoid spurious interrupts, programs must not change a buffer's INTIO setting while the buffer is enabled.

- **Flush DMA Buffers and Status.** DMACx Bit 14 (FLSH). This bit flushes (when set, =1) settings for the corresponding external port FIFO buffer (EPBx).
- **External Port FIFO Buffer Status.** DMACx bit 17-16 (FS). These bits indicate the corresponding FIFO buffer's status.

The HPS and PS bits in the SYSTAT and DMACx registers indicate an external buffer's packing status. These bits are read-only, and the DSP clears these bits when DEN is cleared (changes from 1 to 0). Table 6-13 shows the available settings.

Table 6-13. DSP (PS) and Host (HPS) Packing Status

PS or HPS	Packing Status
00	pack complete (reset value)
01	1st stage pack/unpack
10	2nd stage multi-stage pack/unpack
11	reserved

The FS bits in the DMACx registers indicate an external buffer's FIFO status. These bits are read-only. The DSP clears these bits when DEN is cleared (changes from 1 to 0). Table 6-14 shows the available settings.

Table 6-14. External Port Buffer FIFO Status

FS	FIFO Buffer Status
00	buffer empty
01	buffer-not-full
10	buffer-not-empty
11	buffer full

For transmit (TRAN=1), buffer-not-full means that the buffer has space for one Normal word, and buffer-not-empty means that the buffer has space

for two-or-more Normal words. For receive (TRAN=0), buffer-not-full means that the buffer contains one Normal word, and buffer-not-empty means that the buffer contains two-or-more Normal words. Any type of full status (01, 10, or 11) in receive mode indicates that new (unread) data is in the buffer.

The HPFLSH and FLSH bits in the SYSCON and DMACx registers flush an external buffer's packing status, but these bits work differently. The HPFLSH bit flushes (when set, =1) settings for the direct write FIFO. Flushing these settings clears the HPS status bits in the SYSTAT register, clears (=0) the channel's DMA request counter, and clears (-0) any partially packed words.

By comparison, the FLSH bit flushes (when set, =1) settings for the corresponding external port FIFO buffer (EPBx). Flushing these settings does the following: Clears (=0) the FS and PS status bits, Clears (=0) the FIFO buffer and DMA request counter, Clears (-0) any partially packed words.

When a program sets (=1) either HPFLSH or FLSH, the DSP flushes the settings and clears (=0) the flush bit. There is a two-cycle effect latency in completing the flush operation. DSP programs must not set a buffer's FLSH during the same write that enables the buffer. Also, programs must not set a buffer's FLSH bit while the DMA channel is active. Programs should determine the channel's active status by reading the corresponding bit in the DMASTAT register.



Status does not change on the master DSP during external port DMA until the external portion is completed (i.e., the EPBx buffers are emptied).



If in chain insertion mode (DEN=0, CHEN=1), then channel chaining status will never go to 1. Programs should test channel status to see if it is ready before re-writing the chain pointer (CPx).

The INTIO bit in the DMACx registers support single-word interrupt-driven transfers for each corresponding external port buffer. These non-DMA transfers are available under the following conditions:

- The external port DMA channel's DEN bit is cleared (DMA disabled).
- The external port DMA channel's INTIO bit is set (enabling interrupt-driven I/O).
- The external port DMA channel's buffer is "not empty" on an external read or "not full" on an external write.

Under these conditions, the I/O processor generates that DMA channel's interrupt on the single word transfer to that channel's external port buffer.

Link Port Status

The I/O processor monitors the status of data transfers on the link ports. DMA channel status for the link ports is described in "Using I/O Processor Status" on page 6-53. This section describes link ports specific status features, such as buffer status, buffer control, and single-word interrupt-driven transfers.

Bits in the LCOM and LSRQ registers indicate and control status of link port buffers. Table A-24 on page A-65 lists all the bits in LCOM and Table A-26 on page A-68 lists all the bits in LSRQ. The following bits influence link port buffer status.

- **Link Buffer x Status.** LCOM Bits 1-0, 3-2, 5-4, 7-6, 9-8, 11-10 (LxSTAT). These bits indicate the corresponding buffer's status.
- **Link Buffer x Receive Pack Error Status.** LCOM Bits 26, 27, 28, 29, 30, 31 (LRERRx). These bits indicate the buffer's packing status.

- **Link Port x transmit mask.** LSRQ Bit 4, 6, 8, 10, 12, 14 (LxTM). These bits mask (if set, =1) or unmask (if cleared, =0) the L0TRQ through L5TRQ status bits.
- **Link Port x receive mask.** LSRQ Bit 5, 7, 9, 11, 13, 15 (LxRM). These bits mask (if set, =1) or unmask (if cleared, =0) the L0RRQ through L5RRQ status bits.
- **Link Port x transmit request status (read-only).** LSRQ Bit 20, 22, 24, 26, 28, 30 (LxTRQ). If set (=1), these bits indicate that a link port (0 through 5) is disabled, but has a request to transmit data.
- **Link Port x receive request status (read-only).** LSRQ Bit 21, 23, 25, 27, 29, 31 (LxRRQ). If set (=1), these bits indicate that a link port (0 through 5) is disabled, but has a request to receive data.

The LRERRx bits in the LCOM register indicate a link port buffer's receive packing status. When the buffer is ready to receive and pack a a new word, the DSP clears (=0) LRERRx. If this bit remains set (=1) after the buffer receives a word, a link transfer error (for example, a clock glitch) has occurred. These bits are read-only, and the DSP clears these bits when LxEN is cleared (changes from 1 to 0). Table 6-15 shows the available settings.

Table 6-15. Link Port Buffer Receive Packing Status

LRERRx	Receive Packing Status
0	pack complete (reset value)
1	pack not complete

The LxSTAT bits in the LCOM register indicate a link buffer's FIFO status. When transmitting, these bits indicate when the buffer has space for more data. When receiving, these status bits indicate when the buffer contains new (unread) data. These bits are read-only. The DSP clears these bits

when LxEN is cleared (changes from 1 to 0) and empties the buffer. Table 6-16 shows the available settings.

Table 6-16. Link Port Buffer FIFO Status

LxSTAT	FIFO Buffer Status
00	buffer empty
01	reserved
10	one word
11	buffer full

The LAR register lets programs assign link buffers to link ports. Table 6-10 on page 6-44 shows how the AxLB bits in the LAR register assign link buffers to link ports. Because this mapping allows link ports to be unassigned (no buffer), the I/O processor has an interrupt (LSRQI) to notify programs that an external device has made a read or write request on a disabled link port.

When the an LSRQI interrupt is latched into the IRPTL register, programs use the transmit (LxTRQ) and receive (LxRRQ) request bits in LSRQ register to determine which port has a request. The LSRQ register's bits indicate that:

- For a transmit request (LxTRQ=1), the LSRQI interrupt indicates that the link port (0 through 5) is disabled, but another DSP has requested more data by setting the link port's acknowledge (LxACK=1).
- For a receive request (LxRRQ=1), the LSRQI interrupt indicates that the link port (0 through 5) is disabled, but another DSP has requested to send data by setting the link port's clock (LxCLK=1).

To control sources of link port service requests, the I/O processor lets programs mask these service requests. The LSRQ register provides mask bits for transmit (LxTM) and receive (LxRM) link service requests.

The LxEN bits in the LCTLx register support single-word interrupt-driven transfers for each corresponding link port buffer. These non-DMA transfers are available under these conditions:

- The link port DMA channel's LxDEN bit is cleared (DMA disabled).
- The link port DMA channel's LxEN bit is set (enabling the link buffer).
- The link port DMA channel's buffer is "not empty" on receive or "not full" on transmit.

Under these conditions, the I/O processor generates that DMA channel's interrupt on the single word transfer to that channel's link port buffer.

Serial Port Status

The I/O processor monitors the status of data transfers on the serial ports. DMA channel status for the serial ports is described in "Using I/O Processor Status" on page 6-53. This section describes serial ports specific status features, such as buffer status, transmit buffer underflow, receive buffer overflow, and single-word interrupt-driven transfers.

Bits in the STCTLx and SRCTLx registers indicate and control status of serial port buffers. Table A-27 on page A-71 lists all the bits in STCTLx and Table A-28 on page A-74 lists all the bits in SRCTLx. The following bits influence serial port buffer status:

- **Transmit Underflow Status (sticky, read-only).** STCTLx Bit 29 (TUVF). This bit indicates whether the serial transmit operation has underflowed (if set, =1).
- **Transmit Data Buffer Status (read-only).** STCTLx Bits 31-30 (TXS). These bits indicate the status of the serial port's transmit buffer (TXx).

- **Receive Overflow Status (sticky, read-only).** SRCTLx Bit 29 (ROVF). This bit indicates whether the receive operation has overflowed (if set, =1).
- **Receive Data Buffer Status (read-only).** SRCTLx Bits 31-30 (RXS). These bits indicate the status of the serial port's receive buffer (RXx).

The TXS and RXS bits in the STCTLx and SRCTLx registers indicate a serial port transmit (TXx) or receive (RXx) buffer's FIFO status. Status bits are read-only. Disabling the serial port (setting SPEN=0), clears the status bits and empties the buffer. TXS and RXS may change state if the data is read or written by the processor core while the serial port is disabled.

Table 6-17 shows the available settings.

Table 6-17. Serial Port Transmit and Receive Buffer FIFO Status

TXS or RXS	FIFO Buffer Status
00	buffer empty
01	reserved
10	partially full
11	buffer full

The TUVF and ROVF bits in the STCTLx and SRCTLx registers indicate a serial port transmit underflow or receive overflow to the buffer's FIFO. Status bits are read-only. Disabling the serial port (setting SPEN=0), clears the status bits and empties the buffer. These overflow and underflow bits are sticky; once set, they remain set regardless of buffer status until the serial port is disabled.

The SPEN bit in the STCTLx or SRCTLx register support single-word interrupt-driven transfers for each corresponding serial port transmit or receive buffer. These non-DMA transfers are available under these conditions:

- The serial port DMA channel's SDEN bit is cleared (DMA disabled).
- The serial port DMA channel's SPEN bit is set (enabling the serial port transmit or receive buffer).
- The serial port DMA channel's buffer is "not empty" on receive or "not full" on transmit.

Under these conditions, the I/O processor generates that DMA channel's interrupt on the single word transfer to that channel's link port buffer.

DMA Controller Operation

DMA sequences start in different ways depending on whether DMA chaining is enabled. When chaining is not enabled, only the DMA enable bit (DEN) allows DMA transfers to occur. A DMA sequence starts when one of the following occurs:

- Chaining is disabled and the DMA enable bit (DEN) transitions from low to high.
- Chaining is enabled, DMA is enabled (DEN=1), and the CPx register address field is written with a non-zero value. In this case, TCB chain loading of the channel parameter registers occurs first.
- Chaining is enabled, the CPx register address field is non-zero, and the current DMA sequence finishes. Again, TCB chain loading occurs.

A DMA sequence ends when one of the following occurs:

- The count register decrements to zero (both C and EC for external port channels).
- Chaining is disabled and the channel's DEN bit transitions from high to low. If the DEN bit goes low (=0) and chaining is enabled, the channel enters chain insertion mode and the DMA sequence continues. For more information, see "Inserting a TCB in an Active Chain" on page 6-73.

When a program sets the DEN bit (=1) after a single DMA finishes, the DMA sequence continues from where it left off (for non-chained operations only). To start a new DMA sequence after the current one is finished, a program must first clear the DEN enable bit, write new parameters to the II, IM, and C registers, then set the DEN bit to re-enable DMA. For chained DMA operations, these steps are not necessary. For more information, see "Chaining DMA Processes" on page 6-70.

If a DMA operation completes and the count register is rewritten before the DMA enable bit is cleared, the DMA transfer will restart at the new count.

Once a program starts a DMA process, the process is influenced by two external controls: DMA channel priority and DMA chaining. For more information, see For more information, see "Managing DMA Channel Priority" on page 6-66. or "Chaining DMA Processes" on page 6-70.

Managing DMA Channel Priority

The DMA channels for each of the DSP's I/O ports negotiate channel priority with the I/O processor using an internal DMA request/grant handshake. Each I/O port (link ports, serial ports, and external ports) has one or more DMA channels, with each channel having a single request and a single grant. When a particular channel needs to read or write data

to internal memory, the channel asserts an internal DMA request. The I/O processor prioritizes the request with all other valid DMA requests. When a channel becomes the highest priority requester, the I/O processor asserts the channel's internal DMA grant. In the next clock cycle, the

DMA transfer starts. Figure 6-8 shows the paths for internal DMA requests within the I/O processor.

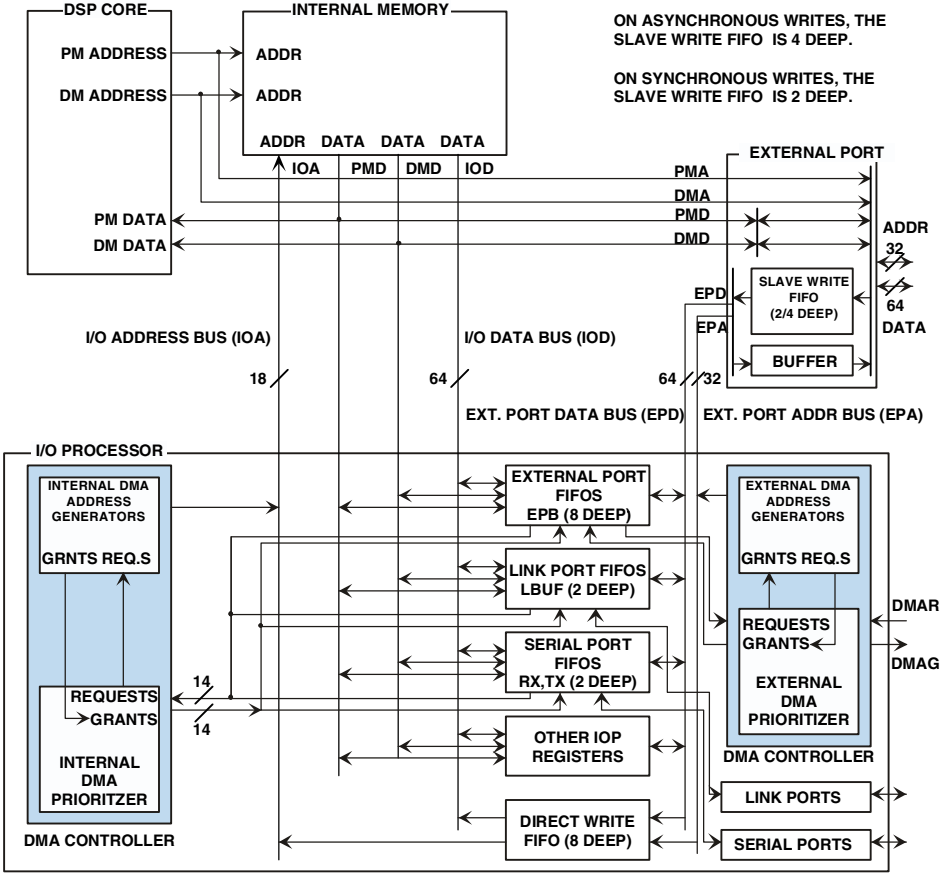


Figure 6-8. I/O Processor Internal Request and Grant Paths

If a DMA channel is disabled (DEN, LxDEN, or SDEN bit =0), the I/O processor does not issue internal DMA grants to that channel, whether or not the channel has data to transfer.

Because more than one DMA channel can make a DMA request in a particular cycle, the I/O processor prioritizes DMA channel service. DMA

channel prioritization determines which channel can use the IOD (I/O Data) bus to access memory. Default DMA channel priority is fixed prioritization by DMA channel type (serial ports, link ports, or external port).

Within the DMA channel types, the serial port DMA channels are always fixed priority, the external port DMA channels may be either fixed or rotated priority, and the link port DMA channels may be either fixed or rotated priority.

Table 6-1 on page 6-11 lists the DMA channels in descending order of priority. For information on programming link port or external port priority modes, see "External Port Channel Priority Modes" on page 6-18 or "Link Port Channel Priority Modes" on page 6-45.

The I/O processor determines which DMA channel has the highest priority internal DMA request during every cycle between each data transfer. Internal DMA channel arbitration differs from external bus arbitration. For more information on external bus arbitration, see "Multiprocessor Bus Arbitration" on page 7-101.

Processor core accesses of I/O processor registers, external direct accesses of internal memory, and TCB chain loading are subject to the same prioritization scheme as the DMA channels. Applying this scheme uniformly prevents I/O bus contention, because these accesses are also performed over the internal I/O bus. TCB chain loading has a higher priority than external port accesses. This TCB priority permits chained serial port DMA, even when the external port is attempting an access in every cycle. For more information, see "Chaining DMA Processes" on page 6-70.

If a DSP has all six link ports enabled and active at the same time, the default priority scheme could hold off external port DMA channels for extended periods of time. Because this hold off could have a significant negative impact on external bus performance, the I/O processor permits rotating DMA channel priority between the link port channel group and external port channel group. For more information on using the PRROT bit

to rotate priority between link ports and the external port, see "Link Port Channel Priority Modes" on page 6-45.

Chaining DMA Processes

DMA chaining lets the I/O processor automatically load DMA parameters and start the next DMA when the current DMA finishes. This feature permits unlimited multiple DMA transfers without processor core intervention. Using chaining, programs can set up multiple DMA operations, and each operation can have different attributes.



The I/O processor responds by auto-initializing the channel's parameter registers with the first TCB and starting the first transfer.

To chain together multiple DMA operations, the I/O processor must load the next Transfer Control Block (DMA parameters) into the DMA parameter registers when the current DMA finishes (DMA count =0). The chain pointer register (CPx) points to the next set of DMA parameters, which are stored in internal memory. This process of loading the TCB into the parameter registers is called TCB chain loading.

The chain pointer register should be cleared first before enabling chaining. Two controls enable chained DMA. Each DMA channel has a chaining enable bit (CHEN) in the channel's control register. When set, the CHEN bit directs the I/O processor to use the CPx register for chained DMA. Programs start the chained DMA by writing a non-zero address to the CPx register, directing the I/O processor to start the DMA with TCB chain loading. Programs can disable chained DMA by writing all zeros to the address field of the CPx register.



Chained DMA operations may only occur within the same channel. The DSP does not support cross-channel chaining.

The CPx register is 19 bits wide, of which the lower 18 bits are the memory address field. Like other I/O processor address registers, the CPx registers

value is offset to match the starting address of internal memory before being used by the I/O processor. On the ADSP-21160 DSP, this offset value is 0x0004 0000.

Bit 18 (the 19th bit) of the CPx register is Program Controlled Interrupts (PCI) bit. If set, the PCI bit enables a DMA channel interrupt to occurs at the completion of the current DMA sequence.



The PCI bit only effects DMA channels that have chaining enabled (CHEN =1). Also, interrupt requests enabled by the PCI bit are maskable with the IMASK register.



Because the PCI bit is not part of the memory address in the CPx register, programs must be careful when writing and reading addresses to and from the register. To prevent errors, programs should mask out the PCI bit (bit 18, the 19th bit) when copying the address in CPx to another address register.

During chained DMA, the channel's General Purpose (GP) register is a useful place to point to the last completed DMA sequence. This practice lets programs determine where the last full (or empty) data buffer is located.

Transfer Control Block (TCB) Chain Loading

During TCB chain loading, the I/O processor loads the DMA channel parameter registers with values retrieved from internal memory. The address in the CPx register points to the highest address of the TCB (containing the IIx parameter). The TCB values reside in consecutive memory locations.

Table 6-18 shows the TCB-to-register loading sequence for the external port, link port, and serial port DMA channels. The parameter order in the table is the order that the I/O processor reads each word of the TCB and loads it into the corresponding register. Programs must set up the TCB in memory in the order shown in Table 6-18, placing the IIx parameter at

the address pointed to by the CPx register of the previous DMA operation of the chain.

Table 6-18. TCB Chain Loading Sequence

Address[1]	External Port	Link Ports and Serial Ports
CPx + 0x0004 0000	IIx	IIx
CPx – 1 + 0x0004 0000	IMx	IMx
CPx – 2 + 0x0004 0000	Cx	Cx (and DAx for 2D DMA)[2]
CPx – 3 + 0x0004 0000	CPx	CPx
CPx – 4 + 0x0004 0000	GPx	GPx
CPx – 5 + 0x0004 0000	EIx	DBx (loaded during 2D DMA only)
CPx – 6 + 0x0004 0000	EMx	LPATH1 (mesh multiproc. links only)[3]
CPx – 7 + 0x0004 0000	ECx	LPATH2 (mesh multiproc. links only)[3]
CPx – 8 + 0x0004 0000	–	LPATH3 (mesh multiproc. links only)[3]

1 An “x” denotes the DMA channel number.
2 The DAx and DBx registers are not loaded during chaining in normal, one-dimensional DMA. In 2D DMA operations, only DBx is loaded. The DAx register is automatically loaded with the same value as the Cx register.
3 The link transmit chain also downloads the LPATH1, LPATH2, and LPATH3 registers when the LMSP bit in the LCOM control register is set, enabling mesh multiprocessing.

A TCB chain load request is prioritized like all other DMA operations. The I/O processor latches a TCB loading request and holds it until the load request has the highest priority. If multiple chaining requests are present, the I/O processor services the TCB registers for the highest priority DMA channel first. A channel which is in the process of chain loading cannot be interrupted by a higher priority channel.

For a list of DMA channels in priority order, see Table 6-1 on page 6-11. For more information on DMA priority, see “Managing DMA Channel Priority” on page 6-66.

Setting Up and Starting The Chain

To setup and initiate a chain of DMA operations, program use the following steps:

1. Set up all TCBs in internal memory.
2. Write to the appropriate DMA control register, setting the DEN DMA enable bit to 1 and the CHEN chaining enable bit to 1.
3. Write the address containing the IIx register value of the first TCB to the CPx register, starting the chain (see Figure 6-9).

The I/O processor responds by autoinitializing the channel's parameter registers with the first TCB and starting the first transfer. When the transfer finishes, the I/O processor begins the next TCB chain load if the current chain pointer address is non-zero. The CPx address points to the next TCB.

The address field of the CPx registers is only 18 bits wide. If a program writes a symbolic address to bit 18 of CPx, there may be a conflict with the PCI bit. Programs should clear the upper bits of the address, then AND in the PCI bit separately if needed.

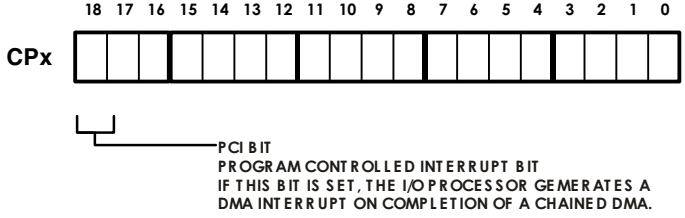

18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CPx
PCI BIT
PROGRAM CONTROLLED INTERRUPT BIT
IF THIS BIT IS SET, THE I/O PROCESSOR GEMERATES A
DMA INTERRUPT ON COMPLETION OF A CHAINED DMA.

Figure 6-9. CPx Register

Inserting a TCB in an Active Chain

It is possible to insert a single DMA operation or another DMA chain within an active DMA chain. Programs may need to perform insertion

when a high priority DMA requires service and cannot wait for the current chain to finish.

When DMA on a channel is disabled (DEN=0) and chaining on the channel is enabled (CHEN=1), the DMA channel is in chain insertion mode. This mode lets a program insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer. Programs should use the following sequence to insert a DMA subchain while another chain is active:

1. Enter chain insertion mode by setting CHEN=1 and DEN=0 in the channel's DMA control register.
2. The DMA interrupt indicates when the current DMA sequence has completed.
3. Write the CPx register value into the CP position of the last TCB in the new chain.
4. Enter chained DMA mode by setting DEN=1 and CHEN=1.
5. Write the start address of the first TCB of the new chain into the CPx register.

Chain insertion mode operates the same as chained DMA mode (DEN=1, CHEN=1), except that when the current DMA transfer ends, automatic chaining is disabled and an interrupt request occurs. This interrupt request is independent of the PCI bit state.



Chain insertion should not be set up as an initial mode of operation. This mode should only be used to insert a DMA within an active DMA chaining operation.

External Port DMA

The DSP support a number of DMA modes for external port DMA. The following sections provide overviews of typical external port DMA processes:

- "Setting up External Port DMA" on page 6-75
- "Bootloading Through The External Port" on page 6-76

Setting up External Port DMA

The method for setting up and starting an external port DMA sequence varies slightly with the selection of transfer and DMA handshake for the channel. For more information on transfer and DMA handshake modes, see "External Port Channel Transfer Modes" on page 6-21 and "External Port Channel Handshake Modes" on page 6-22. For more detailed information on external port DMA features, see "Setting I/O Processor—EPort Modes" on page 6-13.

In general, the following sequence describes a typical external to internal DMA operation where an external device transfers a block of data into the DSP's internal memory:

1. The DSP or host (depends on mode) writes the DMA channel's parameter registers (`IIx`, `IMx`, and `Cx`) and `DMACx` control register, initializing the channel for receive (`TRAN`=0).
2. The DSP or host (depends on mode) sets (=1) the channel's `DEN` bit enabling the DMA process.
3. The external device begins writing data to the `EPBx` buffer (through the external port).
4. Whether or not the DSP signals for this transfer to begin depends on the mode.

5. The EPBx buffer detects data is present and asserts an internal DMA request to the I/O processor.
6. The I/O processor grants the request and performs the internal DMA transfer, emptying the EPBx buffer FIFO.

In general, the following sequence describes a typical internal to external DMA operation where an external device transfers a block of data from the DSP's internal memory:

1. The DSP or host (depends on mode) writes the DMA channel's parameter registers (IIx, IMx, and Cx) and DMACx control register, initializing the channel for transmit (TRAN=1).
2. The DSP or host (depends on mode) sets (=1) the channel's DEN bit enabling the DMA process. Because this is a transmit, setting DEN automatically asserts an internal DMA request to the I/O processor.
3. The I/O processor grants the request and performs the internal DMA transfer, filling the EPBx buffer's FIFO.
4. The external device begins reading data from the EPBx buffer (through the external port).
5. Whether or not the DSP signals for this transfer to begin depends on the mode.
6. The EPBx buffer detects that there is room in the buffer (it is now "partially empty) and asserts another internal DMA request to the I/O processor, continuing the process.

Bootloading Through The External Port

The DSP can boot from an EPROM or host processor through the external port. The DMAC10 control register is specially initialized for booting in each case. Each booting mode packs boot data into 48-bit instructions.

EPROM and host boot use channel 10 of the I/O processor's DMA controller to transfer the instructions to internal memory. For EPROM booting, the DSP reads data from an 8-bit external EPROM. For host booting, the DSP accepts data from a 16- or 32-bit host microprocessor (or other external device).

After the boot process loads 256 words into memory locations 0x40000 through 0x400FF, the DSP begins executing instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. Analog Devices supplies loading routines (Loader Kernels) that can load entire programs. These routines come with the development tools. For more information on Loader Kernels, see the development tools documentation.



It is important to note that DMA channel differences between the ADSP-21160 DSP and previous SHARC DSPs (ADSP-2106x DSPs) introduce some booting differences. Even with these differences, the ADSP-21160 DSP supports the same boot capability and configuration as the ADSP-2106x DSPs.

DMAC default values differ because the ADSP-21160 DSP has additional parameters and different DMA channel assignments. The EPROM and Host boot modes use EPB0, DMA channel 10.

In EPROM booting, the alignment of the 8-bit port differs due to the new 64-bit data path. The ADSP-21160 DSP boots from DATA39-32 instead of DATA23-16 as on the ADSP-2106x DSPs.

For EPROM or host booting the ADSP-21160 DSP, the Program sequencer automatically unmasks the DMA channel 10 channel interrupt, initializing the IMASK register to 0x00008003.

The DSP determines the booting mode at reset from the EBOOT, LBOOT, and $\overline{\text{BMS}}$ pin inputs. When EBOOT=1 and LBOOT=0, the DSP boots from an

EPROM through the External Port and uses $\overline{\text{BMS}}$ as the memory select output. When EBOOT=0, LBOOT=0, and $\overline{\text{BMS}}$ =1, the DSP boots from a host through the External Port.

For a list showing how to select different boot modes, see the Boot Memory Select pin description on "Pin Descriptions" on page 11-3.



When using any of the power-up booting modes, address 0x0004 0004 should not contain a valid instruction since it is not executed during the booting sequence. A NOP or IDLE instruction should be placed at this location.

In EPROM booting through the external port, an 8-bit wide boot EPROM must be connected to data bus pins 39-32 (DATA39-32). The lowest address pins of the DSP should be connected to the EPROM's address lines. The EPROM's chip select should be connected to $\overline{\text{BMS}}$ and its output enable should be connected to $\overline{\text{RDH}}$.

In a multiprocessor system, the $\overline{\text{BMS}}$ output is only driven by the ADSP-21160 DSP bus master. This allows wire-ORing of multiple $\overline{\text{BMS}}$ signals for a single common boot EPROM.



Systems can boot any number of ADSP-21160 DSPs from a single EPROM, using the same code for each processor or differing code for each.

During reset, the DSP's ACK line is internally pulled high with a 2 kΩ equivalent resistor and is held high with an internal keeper latch. It is not necessary to use an external pull-up resistor on the ACK line during booting or at any other time.

When EPROM boot mode is configured, the External Port DMA Channel 10 (DMAC10) becomes active following reset; it is initialized to 0x04A1, which allows external port DMA enable and selects DTYPE for instruction words. The packing mode bits (PMODE) are ignored, BSO is set in SYSCON, and 8-to-48 bit packing is forced with least-significant-word first.

The UBWS and UBAM fields of the WAIT register are initialized to perform asynchronous access and generate seven wait states (eight cycles total) for the EPROM access in unbanked external memory space. (Note that wait states defined for unbanked memory are applied to $\overline{\text{BMS}}$-asserted accesses.)

Table 6-19 shows how the DMA Channel 10 parameter registers are initialized at reset for EPROM booting. The count register (C10) is initialized to 0x0100 for transferring 256 words to internal memory. The external count register (EC10), which is used when external addresses are generated by the DMA controller, is initialized to 0x0600 (that is, 0x0100 words with six bytes per word).

Table 6-19. DMA Channel 10 Parameter Register Initialization For EPROM Booting

Parameter Register	Initialization Value
II10	0x0004 0000
IM10	uninitialized (increment by 1 is automatic)
C10	0x0100 (256 instruction words)
CP10	uninitialized
GP10	uninitialized
EI10	0x0080 0000
EM10	uninitialized (increment by 1 is automatic)
EC10	0x0600 (256 words x 6 bytes/word)

At system start-up, when the DSP's $\overline{\text{RESET}}$ input goes inactive, the following sequence occurs:

1. The DSP goes into an idle state, identical to that caused by the IDLE instruction. The program counter (PC) is set to address 0x0004 0004.
2. The DMA parameter registers for channel 10 are initialized (as shown in Table 6-19).
3. $\overline{\text{BMS}}$ becomes the boot EPROM chip select.
4. 8-bit Master Mode DMA transfers from EPROM to internal memory begin, on the external port data bus lines 39-32.
5. The external address lines (ADDR31-0) start at 0x0080 0000 and increment after each access.
6. The $\overline{\text{RDH}}$ strobe asserts as in a normal memory access, with seven wait states (eight cycles).

The DSP's DMA controller reads the 8-bit EPROM words, packs them into 48-bit instruction words, and transfers them to internal memory until 256 words have been loaded. The EPROM is automatically selected by the $\overline{\text{BMS}}$ pin; other memory select pins are disabled. The DMA external count register (EC10) decrements after each EPROM transfer. When EC10 reaches zero, the following wake-up sequence occurs:

1. The DMA transfers stop.
2. The External Port DMA Channel 10 interrupt (EP0I) is activated.
3. $\overline{\text{BMS}}$ is deactivated and normal external memory selects are activated.
4. The DSP vectors to the EP0I interrupt vector at 0x0004 0050.

At this point the DSP has completed its booting mode and is executing instructions normally. The first instruction at the EP0I interrupt vector

location, address 0x0004 0050, should be an RTI (Return From Interrupt). This process returns execution to the reset routine at location 0x0004 0005 where normal program execution can resume. After reaching this point, a program can write a different service routine at the EP0I vector location 0x0004 0050.

Link Port DMA

The DSP support a number of DMA modes for link port DMA. The following sections provide overviews of typical link port DMA processes:

- "Setting up Link Port DMA" on page 6-81
- "Using Two-Dimensional Link Port DMA" on page 6-83
- "Bootloading Through The Link Port" on page 6-88

Setting up Link Port DMA

The method for setting up and starting an link port DMA sequence varies slightly with the transfer mode for the channel. For more information on DMA transfer modes, see "Link Port Channel Transfer Modes" on page 6-47. For more detailed information on link port DMA features, see "Setting I/O Processor—LPort Modes" on page 6-41.

In general, the following sequence describes a typical external to internal DMA operation where an external device transfers a block of data into the DSP's internal memory using a link port:

1. The DSP or host (depends on mode) assigns the DMA channel's link buffer to a link port using the channel's AxLB bits in the LAR register.
2. The DSP or host (depends on mode) enables the DMA channel's link buffer, setting the buffer's LxEN bit in the channel's LCTLx register. The DSP or host selects a words size (32- or 40/48-bits) using the LxEXT in the channel's LCTLx register.
3. The DSP or host (depends on mode) writes the DMA channel's parameter registers (IIx, IMx, and Cx) and LCTLx control register, initializing the channel for receive (LxTRAN=0).
4. The DSP or host (depends on mode) sets (=1) the channel's LxDEN bit enabling the DMA process.
5. The external device begins writing data to the LBUFx buffer (through the link port).
6. The LBUFx buffer detects data is present and asserts an internal DMA request to the I/O processor.
7. The I/O processor grants the request and performs the internal DMA transfer, emptying the LBUFx buffer FIFO.

In general, the following sequence describes a typical internal to external DMA operation where an external device transfers a block of data from the DSP's internal memory using a link port:

1. The DSP or host (depends on mode) assigns the DMA channel's link buffer to a link port using the channel's AxLB bits in the LAR register.
2. The DSP or host (depends on mode) enables the DMA channel's link buffer, setting the buffer's LxEN bit in the channel's LCTLx register. The DSP or host selects a words size (32- or 40/48-bits) using the LxEXT in the channel's LCTLx register.
3. The DSP or host (depends on mode) writes the DMA channel's parameter registers (IIx, IMx, and Cx) and DMACx control register, initializing the channel for transmit (LxTRAN=1).
4. The DSP or host (depends on mode) sets (=1) the channel's LxDEN bit enabling the DMA process. Because this is a transmit, setting LxDEN automatically asserts an internal DMA request to the I/O processor.
5. The I/O processor grants the request and performs the internal DMA transfer, filling the LBUFx buffer's FIFO.
6. The external device begins reading data from the LBUFx buffer (through the link port).
7. The LBUFx buffer detects that there is room in the buffer (it is now "partially empty) and asserts another internal DMA request to the I/O processor, continuing the process.

Using Two-Dimensional Link Port DMA

Two-dimensional DMA is available on all link port and serial port DMA channels. This DMA mode lets programs DMA data in memory treating the data as an array. Programs can use this row and column data arrange-

ment in DSP algorithms that perform array operations. In this mode, the DMA channel's DAx, DBx, and GPx registers direct data placement operations and the operation of the Cx register changes slightly from single-dimension DMA transfers.

To place a link port or serial port in two-dimensional DMA mode, a program must set the link port channel's LxDMA2D bit in the LCTLx register or the serial port channel's D2DMA bit in the SRCTLx or SRCTLx register.

Figure 6-10 and Figure 6-11 show how registers operate for two-dimensional DMA. These figures also show how the I/O processor places 16- or 32-bit data from a two-dimensional DMA in memory.

In two-dimensional DMA, the I/O processor's parameter registers operate as follows:

- The Index register (IIx) initially holds the first address in the data array. As the DMA progresses, the I/O processor updates IIx to hold the current address by adding the data array X increment after each transfer.
- The Modify register (IMx) holds the data array X (column) increment. The I/O processor uses IMx to modify the current address (IIx) to point to the next element in the data array X dimension (next column of array, not necessarily the next memory column).
- The Dimension-A (DAx) register holds the data array X initial count (column count). At the beginning of each new row, the I/O processor uses DAx to load the Cx register with the number of columns in the data array. For programming convenience, the I/O processor writes the DAx register automatically whenever the processor core writes a count to the Cx register.
- The Count (Cx) register contains the number of data elements left in the current row. At the beginning of each new row, the I/O processor loads Cx from DAx. When Cx decrements to zero, the I/O processor goes to the next row.

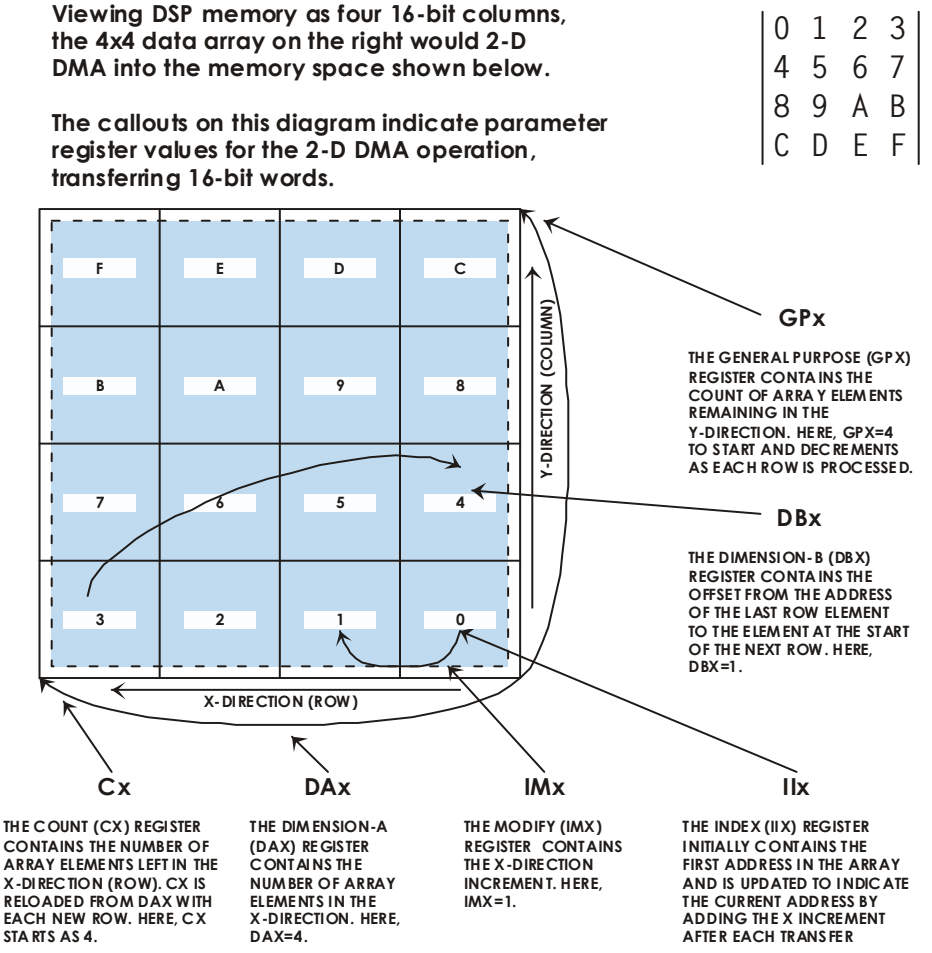

Viewing DSP memory as four 16-bit columns, the 4x4 data array on the right would 2-D DMA into the memory space shown below.
The callouts on this diagram indicate parameter register values for the 2-D DMA operation, transferring 16-bit words.
0 1 2 3
4 5 6 7
8 9 A B
C D E F
F E D C
B A 9 8
7 6 5 4
3 2 1 0
Y-DIRECTION (COLUMN)
X-DIRECTION (ROW)
GPx
THE GENERAL PURPOSE (GPX) REGISTER CONTAINS THE COUNT OF ARRAY ELEMENTS REMAINING IN THE Y-DIRECTION. HERE, GPX=4 TO START AND DECREMENTS AS EACH ROW IS PROCESSED.
DBx
THE DIMENSION-B (DBX) REGISTER CONTAINS THE OFFSET FROM THE ADDRESS OF THE LAST ROW ELEMENT TO THE ELEMENT AT THE START OF THE NEXT ROW. HERE, DBX=1.
Cx
THE COUNT (CX) REGISTER CONTAINS THE NUMBER OF ARRAY ELEMENTS LEFT IN THE X-DIRECTION (ROW). CX IS RELOADED FROM DAX WITH EACH NEW ROW. HERE, CX STARTS AS 4.
DAx
THE DIMENSION-A (DAX) REGISTER CONTAINS THE NUMBER OF ARRAY ELEMENTS IN THE X-DIRECTION. HERE, DAX=4.
IMx
THE MODIFY (IMX) REGISTER CONTAINS THE X-DIRECTION INCREMENT. HERE, IMX=1.
IIx
THE INDEX (IIX) REGISTER INITIALLY CONTAINS THE FIRST ADDRESS IN THE ARRAY AND IS UPDATED TO INDICATE THE CURRENT ADDRESS BY ADDING THE X INCREMENT AFTER EACH TRANSFER

Figure 6-10. Two-Dimensional DMA of a 4x4 Array of 16-bit Words

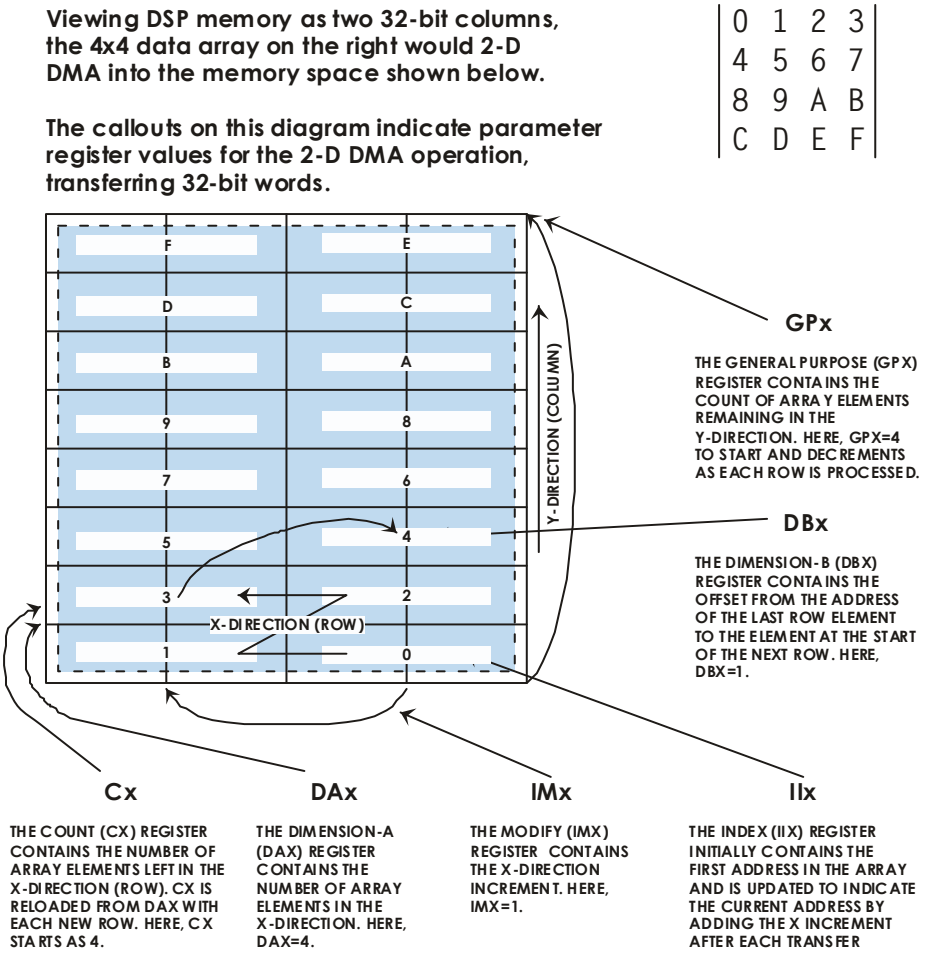

Viewing DSP memory as two 32-bit columns, the 4x4 data array on the right would 2-D DMA into the memory space shown below.
The callouts on this diagram indicate parameter register values for the 2-D DMA operation, transferring 32-bit words.
0 1 2 3
4 5 6 7
8 9 A B
C D E F
F
E
D
C
B
A
9
8
7
6
5
4
3
2
1
0
Y-DIRECTION (COLUMN)
X-DIRECTION (ROW)
GPx
THE GENERAL PURPOSE (GPX) REGISTER CONTAINS THE COUNT OF ARRAY ELEMENTS REMAINING IN THE Y-DIRECTION. HERE, GPX=4 TO START AND DECREMENTS AS EACH ROW IS PROCESSED.
DBx
THE DIMENSION-B (DBX) REGISTER CONTAINS THE OFFSET FROM THE ADDRESS OF THE LAST ROW ELEMENT TO THE ELEMENT AT THE START OF THE NEXT ROW. HERE, DBX=1.
Cx
THE COUNT (CX) REGISTER CONTAINS THE NUMBER OF ARRAY ELEMENTS LEFT IN THE X-DIRECTION (ROW). CX IS RELOADED FROM DAX WITH EACH NEW ROW. HERE, CX STARTS AS 4.
DAx
THE DIMENSION-A (DAX) REGISTER CONTAINS THE NUMBER OF ARRAY ELEMENTS IN THE X-DIRECTION. HERE, DAX=4.
IMx
THE MODIFY (IMX) REGISTER CONTAINS THE X-DIRECTION INCREMENT. HERE, IMX=1.
IIx
THE INDEX (IIX) REGISTER INITIALLY CONTAINS THE FIRST ADDRESS IN THE ARRAY AND IS UPDATED TO INDICATE THE CURRENT ADDRESS BY ADDING THE X INCREMENT AFTER EACH TRANSFER

Figure 6-11. Two-Dimensional DMA of a 4x4 Array of 32-bit Words

- The Dimension-B (DBx) register contains the data array Y increment (row increment). To start a new row, the I/O processor adds the offset in DBx to the current address to point to the next element in the Y dimension (first location in next row).
- The General Purpose (GPx) register contains the data array Y Count. Initially, the GPx contains the number of data elements in the Y dimension (number of rows). The I/O processor decrements this value each time the X count register reaches zero. When Y Count reaches zero, the two-dimensional DMA is finished. To do one-dimensional DMA transfers in two-dimensional DMA mode, programs must set the Y Count GPx to one.



Because a number of parameter registers must update, two DMA cycles are required for a row change.

For two-dimensional DMA transfers, these register operations occur in the following processor order:

1. During the first DMA cycle, the I/O processor performs the following steps:
    a. Outputs the current address from the IIx register and starts a DMA memory cycle
    b. Adds the X Increment value from the IMx register to the current address in the IIx register

c. Decrements the X Count in the Cx register

d. Checks whether the X Count has decremented to zero; if so, performs the second DMA cycle

2. During the second DMA cycle, the I/O processor performs the following steps:

a. Restores the X Count in the Cx register from the DAx register

b. Adds the Y Increment value in the DBx register to the current address in the IIx register

c. Decrements the Y Count in the GPx register

d. Checks whether the Y Count has decremented to zero; if so, the DMA sequence is finished (the channel becomes inactive)



If a program loads the X Count (Cx) register or Y Count (GPx) with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count occurs because the I/O processor starts the first transfer before the testing the count value. The only way to disable a DMA channel is to clear its DMA enable bit.

For more information, see "External Port Channel Transfer Modes" on page 6-21, "Link Port Channel Transfer Modes" on page 6-47, or "Serial Port Channel Transfer Modes" on page 6-52.

Bootloading Through The Link Port

One of the DSP's booting mode is booting the DSP through the link port. Link port booting uses DMA channel 8 of the I/O processor to transfer the instructions to internal memory. In this boot mode, the DSP receives 4-bit wide data in link buffer 4.

After the boot process loads 256 words into memory locations 0x40000 through 0x400FF, the DSP begins executing instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. Analog Devices supplies loading routines (Loader Kernels) that load an entire program through the selected port. These routines come with the development tools. For more information on Loader Kernels, see the development tools documentation.



DMA channel differences between the ADSP-21160 DSP and previous SHARC DSPs (ADSP-2106x DSPs) introduce some booting differences. Even with these differences, the ADSP-21160 DSPs supports the same boot capability and configuration as the ADSP-2106x DSPs. For link booting the ADSP-21160 DSP, the Program sequencer automatically unmasks the DMA channel 8 interrupt, initializing the LIRPTL register to 0x00100000 and IMASK register to 0x00004003.

The DSP determines the booting mode at reset from the EBOOT, LBOOT, and $\overline{\text{BMS}}$ pin inputs. When EBOOT=0, LBOOT=1, and $\overline{\text{BMS}}$=1, the DSP boots through the Link Port. For a list showing how to select different boot modes, see the Boot Memory Select pin description on Table 11-1 on page 11-3.



When using any of the power-up booting modes, address 0x0004 0004 should not contain a valid instruction since it is not executed during the booting sequence. A NOP or IDLE instruction should be placed at this location.

In Link Port Booting, the DSP gets boot data from another DSP's link port or four bit wide external device after system powerup.

The external device must provide a clock signal to the link port assigned to link buffer 4. The clock can be any frequency, up to a maximum of the DSP clock frequency. The clock's falling edges strobe the data into the

link port. The most significant 4-bit nibble of the 48-bit instruction must be downloaded first.

Table 6-20 shows how the DMA Channel 8 parameter registers are initialized at reset for Link Port booting. The count register (C8) is initialized to 0x0100 for transferring 256 words to internal memory. The LCTL and LCOM link port control registers are overridden during link port booting to allow link buffer 4 to receive 48-bit data.

Table 6-20. DMA Channel 8 Parameter Register Initialization For Link Port Booting

Parameter Register	Initialization Value
II8	0x0004 0000
IM8	uninitialized (increment by 1 is automatic)
C8	0x0100 (256 instruction words)
CP8	uninitialized
GP8	uninitialized
DA	uninitialized
DB	uninitialized

In systems where multiple DSPs are not connected by the parallel external bus, booting can be accomplished from a single source through the link ports. To simultaneously boot all of the DSPs, a parallel common connection should be made to Link Buffer 4 on each of the processors. If only a daisy chain connection exists between the processors' link ports, then each DSP can boot the next one in turn. Link Buffer 4 must always be used for booting.

Serial Port DMA

The DSP support a number of DMA modes for link port DMA. The following sections provide overviews of typical serial port DMA processes:

- "Setting up Serial Port DMA" on page 6-91
- "Using Two-Dimensional Serial Port DMA" on page 6-93

Setting up Serial Port DMA

The method for setting up and starting an serial port DMA sequence varies slightly with the transfer mode for the channel. For more information on DMA transfer modes, see "Serial Port Channel Transfer Modes" on page 6-52. For more detailed information on serial port DMA features, see "Setting I/O Processor—SPort Modes" on page 6-49.

In general, the following sequence describes a typical external to internal DMA operation where an external device transfers a block of data into the DSP's internal memory using a serial port:

1. The DSP or host (depends on mode) enables the DMA channel's serial port, setting the port's SPEN bit in the port's SRCTLx register. The DSP or host selects a words size using the DTYPE in the ports's SRCTLx register.
2. The DSP or host (depends on mode) writes the DMA channel's parameter registers (IIx, IMx, and Cx) and SRCTLx control register, initializing the channel for receive.
3. The DSP or host (depends on mode) sets (=1) the channel's SDEN bit enabling the DMA process.
4. The external device begins writing data to the RXx buffer (through the serial port).

5. The RXx buffer detects data is present and asserts an internal DMA request to the I/O processor.
6. The I/O processor grants the request and performs the internal DMA transfer, emptying the RXx buffer.

In general, the following sequence describes a typical internal to external DMA operation where an external device transfers a block of data from the DSP's internal memory using a serial port:

1. The DSP or host (depends on mode) enables the DMA channel's serial port, setting the port's SPEN bit in the port's STCTLx register. The DSP or host selects a words size using the DTYPE in the port's STCTLx register.
2. The DSP or host (depends on mode) writes the DMA channel's parameter registers (IIx, IMx, and Cx) and STCTLx control register, initializing the channel for transmit.
3. The DSP or host (depends on mode) sets (=1) the channel's SDEN bit enabling the DMA process. Because this is a transmit, setting SDEN automatically asserts an internal DMA request to the I/O processor.
4. The I/O processor grants the request and performs the internal DMA transfer, filling the TXx buffer.
5. The external device begins reading data from the TXx buffer (through the serial port).
6. The TXx buffer detects that there is room in the buffer (it is now "partially empty) and asserts another internal DMA request to the I/O processor, continuing the process.

Using Two-Dimensional Serial Port DMA

Two-dimensional DMA is available on all link port and serial port DMA channels. This DMA mode lets programs DMA data in memory treating the data as an array. For more information, see "Using Two-Dimensional Link Port DMA" on page 6-83.

Optimizing DMA Throughput

This section discusses overall DMA throughput when several DMA channels are trying to access internal or external memory at the same time. Table 6-21 on page 6-94 summarizes the advantages of different system configurations.

Internal Memory DMA

The DMA channels arbitrate for access to the DSP's internal memory. The DMA controller determines, on a cycle-by-cycle basis, which channel is allowed access to the internal I/O bus and consequently which channel will read or write to internal memory. The priority order of the DMA channels appears in Table 6-1 on page 6-11.

Each DMA transfer takes one clock cycle even when different DMA channels are being allowed access on sequential cycles; i.e. there is no overall throughput loss in switching between channels. Thus, four link port DMA channels, each transferring one byte per cycle, would have the same I/O transfer rate as one external port DMA channel transferring data to internal memory on every cycle. Any combination of link ports, serial ports, and external port transfers has the same maximum transfer rate.

External Memory DMA

When the DMA transfer is between DSP internal memory and external memory, the external memory may have one or more wait states. Figure 6-12 shows an example DMA hardware interface.

Table 6-21. Configurations for DSP—DSP (ADSP-2116x) DMA

DSP Configuration (Data Source)	DSP Configuration (Data Destination)	C/T[1]	Advantages, Disadvantages
Bus Master DMA Master Mode (MASTER= 1) TRAN=1, EIx=address of EPBx buffer in destination, EMx= 0	Bus Slave DMA Slave Mode (MASTER= 0), TRAN= 0	1	Advantage: Destination automatically generates interrupt upon completion. Disadvantage: DMA must be programmed on both source and destination.
Bus Master DMA Master Mode (MASTER= 1) TRAN=1, EIx=MMS address in destination[2], EMx=1	Bus Slave Direct Write	1	Advantage: No programming required for destination. Disadvantage: No interrupt generated upon completion—source must issue vector interrupt to destination.
Bus Slave DMA Slave Mode (MASTER= 0), TRAN= 1	Bus Master DMA Master Mode (MASTER= 1), TRAN=0, EIx=address of EPBx buffer in source, EMx=0	3[3]	Advantage: Source automatically generates interrupt upon completion. Disadvantages: Slower throughput. DMA must be programmed on both source and destination.
Bus Slave Direct Read	Bus Master DMA Master Mode (MASTER= 1), TRAN=0, EIx=MMS address in source[2], EMx=1	4[3]	Advantage: No programming required for source. Disadvantages: Slowest throughput. No interrupt generated upon completion— destination must issue vector interrupt to source.

1 C/T is throughput in cycles/transfer.
2 MMS= Multiprocessor Memory Space
3 Maximum burst throughput: 3-2-2-2, 4-2-2-2

External memory wait states, however, do not reduce the overall internal DMA transfer rate if other channels have data available to transfer. In other words, the DSP's internal I/O data bus will not be held up by an incomplete external transfer.

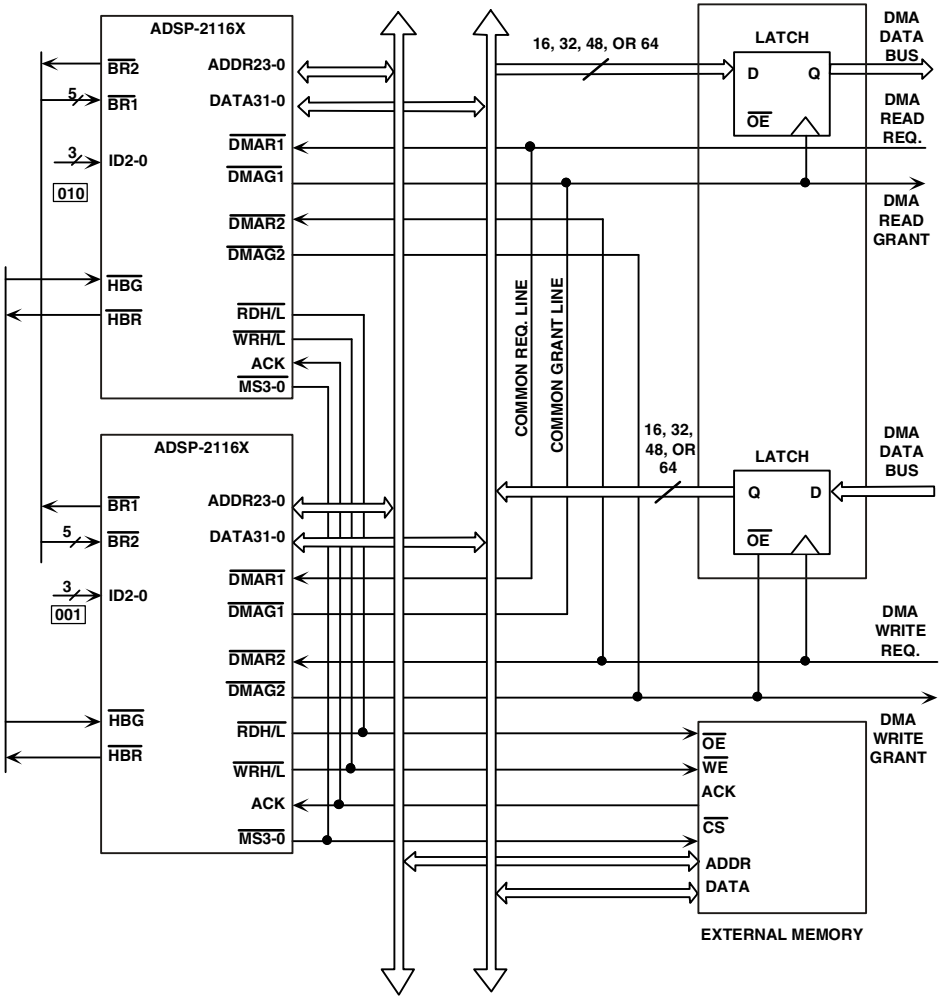


Figure 6-12. Example DMA Hardware Interface

Notes on Figure 6-12:

Because $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ are tied together, only one of the DSPs may have DMA enabled at a time.

- $\overline{\text{DMAGx}}$ is only driven by the DSP bus master.
- The DMA Write Grant signal can be the combination of $\overline{\text{RDH/L}}$ and $\overline{\text{MS3-0}}$ instead of $\overline{\text{DMAG2}}$ if paced master mode is used.
- The DMA Read Grant signal can be the combination of $\overline{\text{WRH/L}}$ and $\overline{\text{MS3-0}}$ instead of $\overline{\text{DMAG1}}$ if paced master mode is used.
- DMA transfers may be to either DSP or to external memory (in external handshake mode).

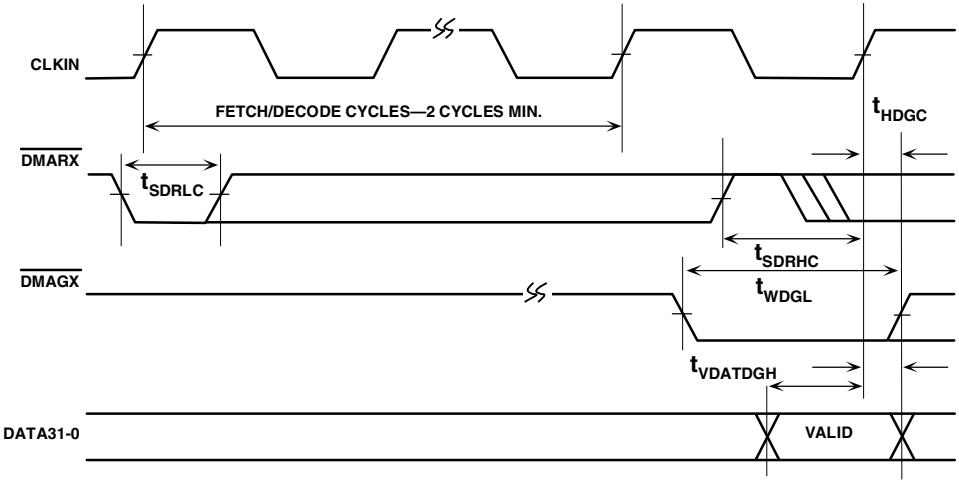


Figure 6-13. $\overline{\text{DMAR}}$ and $\overline{\text{DMAG}}$ Timing

Notes on Figure 6-13:

- $\overline{\text{DMARx}}$ setup times relate to the use of the signal in that cycle by the DSP. DMA requests may be asserted asynchronously to CLKIN.
- $\overline{\text{DMAGx}}$ drives DATA63-0 if DSP is receiving. $\overline{\text{DMAGx}}$ latches DATA63-0 if DSP is transmitting.

When data is to be transferred from internal to external memory, the internal memory data is first placed in the external port's EPBx buffer by the DMA controller; the external memory access is then begun independently. (Likewise for external-to-internal DMA, the internal DMA request will not be made until the external memory data is in the EPBx buffer.) In both cases, the external DMA address generator—the EI and EM parameter registers—maintains the external address until the data transfer is completed. The internal and external address generators of a DMA channel are decoupled and operate independently.

When EXTERN mode DMA transfers occur between an external device and external memory, no internal resources of the DSP are utilized and internal DMA throughput is not affected.

System-Level Considerations

Slave mode DMA is useful in systems with a host processor because it allows the host to access any DSP internal memory location while limiting the address space the host must recognize—only the address space of the DSP's I/O processor registers. Slave mode DMA is also useful for DSP-to-DSP DMA transfers.

Slave mode DMA has one drawback when interfacing to a slow host—the fact that the external bus is held up during the transfer (whether initiated by the DSP or the host) and no other transactions can proceed. To overcome this, the handshake DMA mode may be used.

In Handshake mode, the host does not have to master the bus in order to make a DMA request, nor does the DSP (in master mode) have to wait on the bus for the transfer to complete. Instead, the host asserts the $\overline{\text{DMARx}}$ pin. When the DSP is ready to make the transfer, it can complete it in one bus cycle. For more information, see “Handshake Mode” on page 6-33.

7 EXTERNAL PORT

The DSP's external port extends the DSP's address and data buses off-chip. Using these buses and external control lines, systems can interface the DSP with external memory, 16- or 32-bit host processors, and other DSPs. Because many of the external port operations relate to external memory accessing or I/O processing, this chapter refers to the memory and I/O processor chapters ("Memory" and "I/O Processor") frequently.

Overview

This chapter describes connection and timing issues for the external port. The main sections of this chapter describe the interfaces that are available through the external port. These interfaces include:

- "External Memory Interface" on page 7-3
- "Host Processor Interface" on page 7-51
- "Multiprocessor (DSPs) Interface" on page 7-94

Data alignment through the external port is identical for these interfaces. Figure 7-1 on page 7-2 shows the external port's data alignment.

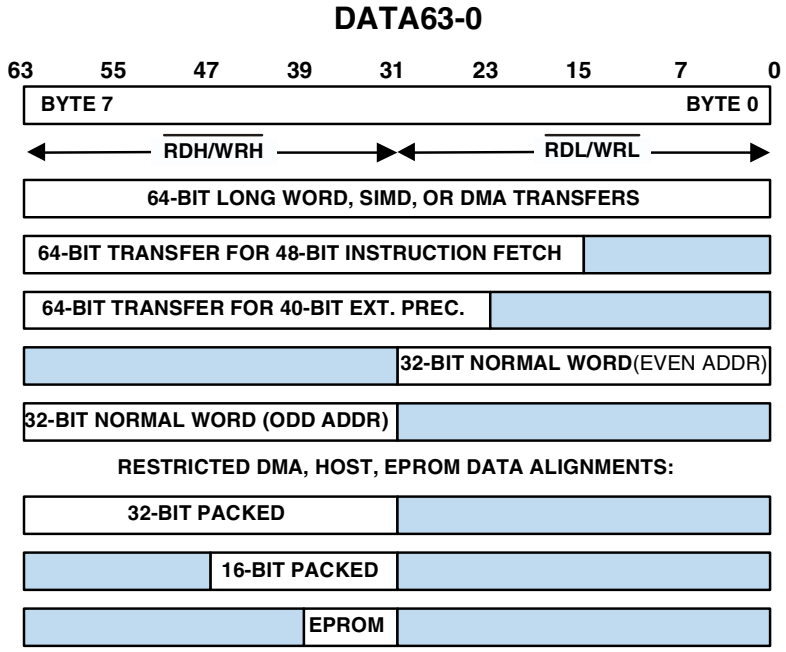


Figure 7-1. External Port Word Alignment

Setting External Port Modes

The SYSCON, WAIT, and DMACx registers control the external port operating mode. Table A-17 on page A-45 lists all the bits in SYSCON, Table A-19 on page A-49 lists all the bits in WAIT, and Table A-21 on page A-55 lists all the bits in DMACx. For information about setting up memory access modes (synchronous versus asynchronous interface), see "Setting Data Access Modes" on page 5-28.

For information on setting DMA through the external port, see "Setting I/O Processor—EPort Modes" on page 6-13. For information on using external port interrupts, see "Using I/O Processor Status" on page 6-53.



There is a 3:1 conflict resolution ratio at the external port interface (three internal buses to one external bus) in addition to the 2:1 or greater clock ratio between the DSP's internal clock and the external system clock. Systems that fetch instructions or data through the external port must tolerate at least one cycle—and possibly many additional cycles—of latency.

External Memory Interface

In addition to its on-chip SRAM, the DSP provides addressing of up to 4 gigawords of off-chip memory through its external port. This external address space includes multiprocessor memory space—the on-chip memory of all other DSPs connected in a multiprocessor system—as well as external memory space—the region for standard addressing of off-chip memory.

Figure 7-2 shows how the buses and control signals extend off-chip, connecting to external memory.

The DSP's memory control signals permit direct connection to fast static RAM devices. Memory mapped peripherals and slower memories can also connect to the DSP using a user-defined combination of programmable waitstates and hardware acknowledge signals.

External memory can hold instructions and data. The external data bus (DATA63-0) must be 64 bits wide to transfer 48-bit instructions and 40-bit extended-precision floating-point data without data packing. If external memory contains only data or packed instructions for transfer by DMA, the external data bus width can be either 16 or 32 bits wide. In a 16- or 32-bit bus system, the DSP's on-chip I/O processor unpacks incoming

data and packs outgoing data. Figure 7-1 shows how the DSP transfers different data word sizes over the external port.

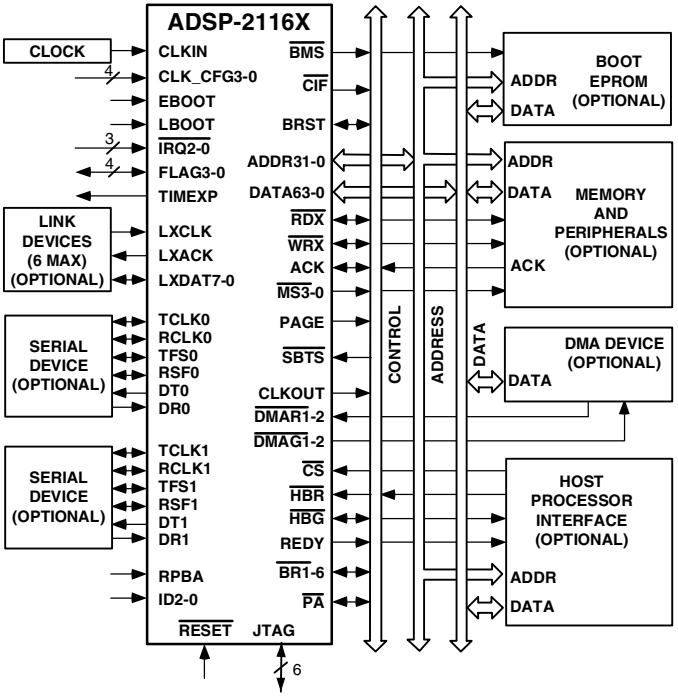


Figure 7-2. ADSP-21535 Processor System

Table 7-1 defines the DSP pins used for interfacing to external memory.

Table 7-1. External Memory Interface Signals

Pin	Type	Function
ADDR 31-0	I/O/T	External Bus Address. The DSP outputs addresses for external memory and peripherals on these pins. In a multiprocessor system, the bus master outputs addresses for read/writes of the internal memory or IOP registers of other DSPs. The DSP inputs addresses when a host processor or multiprocessing bus master is reading or writing its internal memory or I/O processor registers.
DATA 63-0	I/O/T	External Bus Data. The DSP inputs and outputs data and instructions on these pins. 32-bit single-precision floating-point data and 32-bit fixed-point data is transferred over bits 63-32 or 31-0 of the bus. 40-bit extended-precision floating-point data is transferred over bits 63-24 of the bus. 16-bit short word data is transferred over bits 47-32 of the bus. Pull-up resistors on unused DATA pins are not necessary. In asynchronous access mode, read data is sampled by the rising edge of the read strobe (DATA 63-32 sampled with $\overline{\text{RDH}}$, DATA31-0 sampled with $\overline{\text{RDL}}$). On write operations, the data is driven from rising edge of CLKIN, before the write strobes are asserted.
$\overline{\text{MS3-0}}$	O/T	Memory Select Lines. These lines are asserted (low) as chip selects for the corresponding banks of external memory. Memory bank size must be defined in the DSP's system control register (SYSCON). The $\overline{\text{MS3-0}}$ lines are decoded memory address lines that change at the same time as the other address lines. When no external memory access is occurring the $\overline{\text{MS3-0}}$ lines are inactive. In asynchronous access mode, the $\overline{\text{MS}}$x signal is asserted for the whole access. In synchronous access mode, the $\overline{\text{MS}}$x signal is only asserted until ACK is sampled asserted. $\overline{\text{MS0}}$ can be used with the PAGE signal to implement a bank of DRAM memory (Bank 0). In a multiprocessing system the $\overline{\text{MS3-0}}$ lines are output by the bus master. Unlike previous SHARC DSPs, strobe assertion for conditional instructions occurs only when the instruction condition code evaluates as true.
CLKOUT	O/T	Synchronous output clock. Output clock signal at same rate as CLKIN. Output by current bus master.
I (Input), S (Synchronous), o/d (Open Drain), O (Output), A (Asynchronous), a/d (Active Drive), T (Three-state, when $\overline{\text{SBTS}}$ or $\overline{\text{HBR}}$ is asserted, or when the DSP is a bus slave)		

Table 7-1. External Memory Interface Signals (Cont'd)

Pin	Type	Function
PAGE	O/T	DRAM Page Boundary. The DSP asserts this pin to signal that an external DRAM page boundary has been crossed. DRAM page size must be defined in the DSP's memory control register (WAIT). DRAM can only be implemented in external memory Bank 0; the PAGE signal can only be activated for Bank 0 accesses. In a multiprocessing system, PAGE is output by the bus master.
$\overline{\text{RDH/L}}$	I/O/T	Read High, and Read Low Strobes. $\overline{\text{RDH}}$ indicates that a read of the high word of the data bus (DATA63-32) is in progress. $\overline{\text{RDL}}$ indicates that a read of the low word of the data bus (DATA31-0) is in progress. As a master, the DSP asserts the strobe after the ADDR31-0 and $\overline{\text{MS3-0}}$ assert, unless the following bus operation is to the same bank or multiprocessor memory and asserts the same strobe. Timing of the deassertion of the strobe depends upon the access mode. In asynchronous access mode, the strobe is deasserted before the rising edge of CLKIN. For an access to a bank in synchronous access mode, the strobe is deasserted on the rising edge of CLKIN. As a slave, the DSP samples this input to determine the type of bus operation, as well as the size and data alignment for the transfer.
$\overline{\text{WRH/L}}$	I/O/T	Write High, and Write Low Strobes. $\overline{\text{WRH}}$ indicates that a write on the high word of the data bus (DATA63-32) is in progress. $\overline{\text{WRL}}$ indicates that a write on the low word of the data bus (DATA31-0) is in progress. As a master, the DSP asserts the strobe after the ADDR31-0 and $\overline{\text{MS3-0}}$ assert, unless the following bus operation is to the same bank or multiprocessor memory and asserts the same strobe. Timing of the deassertion of the strobe depends upon the access mode. In asynchronous access mode, the strobe is deasserted before the rising edge of CLKIN. For an access to a bank in synchronous access mode, the strobe is deasserted on the rising edge of CLKIN. As a slave, the DSP samples this input to determine the type of bus operation, as well as the size and data alignment for the transfer.

I (Input), S (Synchronous), o/d (Open Drain), O (Output), A (Asynchronous), a/d (Active Drive), T (Three-state, when $\overline{\text{SBTS}}$ or $\overline{\text{HBR}}$ is asserted, or when the DSP is a bus slave)

Table 7-1. External Memory Interface Signals (Cont'd)

Pin	Type	Function
$\overline{\text{CIF}}$	O/T	Core Instruction Fetch. As a master, the DSP asserts (low) this output when the program sequencer of the DSP is making an off-chip instruction fetch (read) only. The address generated for this request is a 48-bit instruction pointer. If the instruction fetch is to an address in one of the external memory banks, the $\overline{\text{MSx}}$ output for that bank is also asserted. This output has timing similar to the $\overline{\text{MS3-0}}$signals.
BRST	I/O/T	Burst Transfer. This signal is asserted (high) by a bus master, to indicate that the current bus read or write transfers a block of data to contiguous, incrementing, 64-bit aligned addresses, over multiple cycles. Each individual data transfer requires an acknowledgment (ACK assertion) from the slave addressed by the transfer. BRST is asserted as an output by the DSP bus master in the cycle after the first cycle in which ACK is sampled asserted. As a synchronous slave, the DSP samples the BRST input to determine if a burst read transfer is in progress. The DSP slave does not support burst write transfers. When interfacing to SBSRAM gluelessly, this output should be connected to the $\overline{\text{ADSC}}$ input of the SBSRAMs (not $\overline{\text{ADV}}$).
ACK	I/O/S	Memory Acknowledge. External devices can deassert ACK (low) to add waitstates to an external memory access (including individual transfers within a burst access). ACK is used by I/O devices, memory controllers, or other peripherals to hold off completion of an external memory access. As a bus master, the DSP samples this input. In asynchronous access mode, ACK is not sampled until the programmed number of waitstates for the access have been counted. For an access to a bank in synchronous access mode, ACK is sampled each CLKIN cycle even during programmed waitstate count. The DSP has a keeper latch on its ACK pin that maintains the input at the level it was last driven. ACK must be sampled high by the DSP before it asserts the strobe(s) for a bus operation. Slaves must assert ACK before three-stating this signal. As a slave, the DSP deasserts ACK as an output, to add waitstates to a synchronous access of its internal memory or IOP register space.
I (Input), S (Synchronous), o/d (Open Drain), O (Output), A (Asynchronous), a/d (Active Drive), T (Three-state, when $\overline{\text{SBTS}}$ or $\overline{\text{HBR}}$ is asserted, or when the DSP is a bus slave)		



For maximum flexibility when interfacing the DSP to 32-bit wide memory, connect the memory's data lines to the DSP's DATA63-32 pins; do not connect the A0 pin. This alignment permits more packing options and lets supports easier DMA to the external memory. In DMA accesses to such memory, the DMA uses a stride of two.

Figure 7-1 also shows how the DSP stores unpacked data and instructions in the 64-bit wide external memory. The external memory map is organized such that consecutive addresses access adjacent 32-bit memory locations. For off-chip instruction fetches, the program sequencer accesses adjacent 48-bit wide memory locations.



The ADSP-21535 external memory interface differs from previous SHARC DSPs. Compared to previous SHARC DSPs, the interface has added signals that support burst transfers and the 64-bit data bus. The synchronous interface delivers greater performance, while the asynchronous interface remains similar to previous SHARC DSPs. The external interface provides glueless support for many asynchronous and/or synchronous devices, including other DSPs. The DSP's burst transfer protocol supports Synchronous Burst SRAMs (SBSRAMs).

Because the memory sub-system uses a 64-bit wide data bus, the DSP has high and low read and write strobes ($\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, $\overline{\text{WRL}}$) to mask and enable 32-bit normal word lanes on the DATA63-0 bus. Note that the least significant bit, ADDR0, of the ADDR31-0 bus may be disregarded during DSP external memory space accesses of 32-bit locations ($\overline{\text{CIF}}$ deasserted), as this information is redundant with the strobes. For more information on packing modes in which the DSP only uses the $\overline{\text{RDH}}$ and $\overline{\text{WRH}}$ pins for accesses, see Table 6-2 on page 6-7.



Systems require the least significant address bit to support off-chip instruction execution by the core (Core Instruction Fetch, $\overline{\text{CIF}}$, asserted), DMA packing modes (including EPROM booting), and host-DSP accesses.

External memory can hold both instructions and data. The external memory must support the full width of the data bus (DATA63-0) to achieve maximum performance. If the DSP DAGs generate external accesses to Long word data (including 48-bit instructions or 40-bit Extended Precision Normal word data) or if the DSP accesses external memory while in SIMD mode, the system must implement the full 64-bit external data bus. Also, the system must support the full 64-bit external data bus if the DSP makes burst DMA transfers.



The ADSP-21535 DSP does not support direct data transfers of 48-bit instructions or 40-bit extended precision data to or from external memory.

For example:

```
dm(0x800100) = r0;       // moves 32 MSBs of r0
dm(0x800100) = r0 (LW); // moves 32 bits from r0
                         // and 32-bits from r1
```

To move instructions or 40-bit extended precision data to or from external memory, programs should use the PX register as an intermediate 64-bit holding register. Also, programs can use the I/O processor to transfer this data through an EPBx FIFO.

For example:

```
dm(0x800100) = px;       //moves 48 MSBs of px
```



The ADSP-21535's external PM address bus is 32 bits wide. The DSP's DM address, PM address, and I/O processor can address the entire 4-gigaword external memory space. The ADSP-21535's program sequencer, like previous SHARC DSPs, only can address the low 24-bits of address space.

Banked External Memory

The DSP divides external memory into four equal-size, programmable banks. By mapping peripherals into different banks, systems can accommodate I/O devices with different timing requirements. For information on configuring these memory banks for waitstates and synchronous or asynchronous access modes, see "Setting Data Access Modes" on page 5-28.



On the ADSP-21535 DSP, Bank 0 starts at address 0x0080 0000 in external memory and is followed in order by Banks 1, 2, and 3. When the DSP generates an address located within one of the four banks, the DSP asserts the corresponding memory select line, $\overline{\text{MS3-0}}$.

The $\overline{\text{MS3-0}}$ outputs serve as chip selects for memories or other external devices, eliminating the need for external decoding logic. $\overline{\text{MS0}}$ provides a select line for an optional bank of DRAM memory, when used in combination with the PAGE signal. For more information, see "DRAM Page Boundary Detection" on page 7-15.

The $\overline{\text{MS3-0}}$ lines are decoded memory address lines that change at the same time as the other address lines. When no external memory access is occurring, the $\overline{\text{MS3-0}}$ lines are inactive.



Unlike previous SHARC DSPs, strobe assertion for conditional instructions occurs only when the instruction condition code evaluates as true.

Unbanked External Memory

The region of external memory above Banks 0-3 is called unbanked external memory space. No $\overline{\text{MSx}}$ memory select line is asserted for accesses in this address space. For information on configuring this unbanked memory for waitstates and synchronous or asynchronous access modes, see "Setting Data Access Modes" on page 5-28.

Boot Memory

Most often, the DSP only asserts the $\overline{\text{BMS}}$ memory select line when the DSP is reading from a boot EPROM. This line allows access to a separate external memory space for booting. Unbanked memory waitstates and mode are applied to $\overline{\text{BMS}}$-selected accesses.

The $\overline{\text{BMS}}$ output is only driven by the DSP bus master. For more information on booting, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

It is also possible to write to boot memory using $\overline{\text{BMS}}$. For more information, see "Using Boot Memory" on page 5-29.

Idle Cycle

A bus idle cycle is an inactive bus cycle that the DSP automatically generates to avoid data bus driver conflicts. Such a conflict can occur when a device with a long output disable time continues to drive after $\overline{\text{RDH/L}}$ is deasserted while another device begins driving on the following cycle. Idle cycles are also required to provide time for a slave in one bank to three-state its ACK driver, before the slave in the next bank enables its ACK driver in the synchronous access modes.

Figure 7-3 shows idle cycle insertion between a synchronous read and a zero-wait, synchronous write in cycle 3.

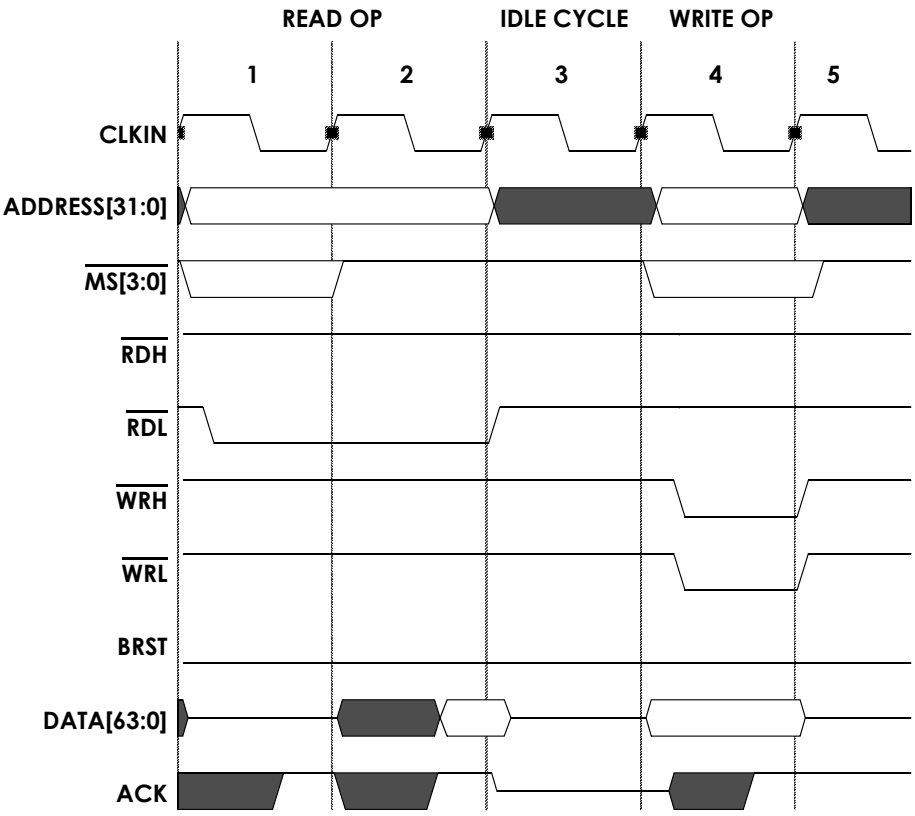


Figure 7-3. Idle Cycle Example

To avoid this conflict, the DSP generates an idle cycle in the following cases.

- On a transition from a read operation to a write operation in the same bank.
- On a transition from one bank, or multiprocessor memory `ID` space to any other bank or multiprocessor slave `ID` space, independent of access mode.

Unlike previous SHARC DSPs, the ADSP-21535 DSP does not support idle cycle insertion on a page boundary crossing.

Data Hold Cycle

The data hold cycle is another configurable memory access feature for adding cycles much like waitstates, as discussed in "Setting Data Access Modes" on page 5-28. A hold time cycle is an inactive bus cycle that the DSP automatically generates at the end of a read or write to allow a longer hold time for address and data. The address, data (if a write), and bank select (if in banked external memory) remain unchanged and are driven for one cycle after the read or write strobes are deasserted. The DSP inserts the data hold cycle only in asynchronous mode and only if the number of programmed waitstates code is 010–111.

Figure 7-4 demonstrates a hold time cycle appended to an asynchronous write access (`EBxWS`=011).

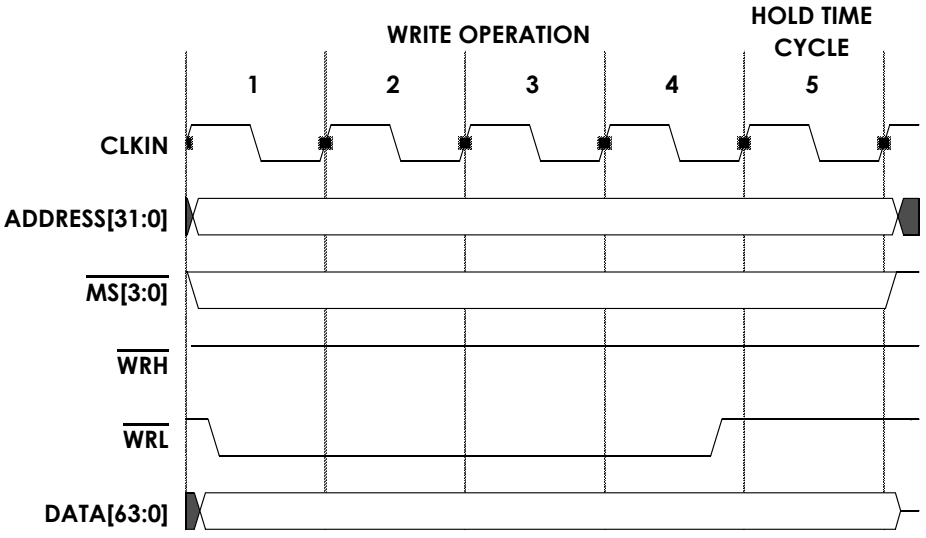


Figure 7-4. Hold Time Cycle Example

The ADSP-21535 DSP does not append an Idle cycle after a Hold cycle.

Multiprocessor Memory Space Waitstates and Acknowledge

Multiprocessor memory space uses only the synchronous transfer protocols, using the zero-waitstate access for writes and a minimum 1-waitstate access for reads. Slave DSPs deassert ACK if more access time is required. DMA burst transfers are only defined for direct read access of a DSP slave's internal memory and reads from the external port buffers (EPBx). For more information, see "Multiprocessor (DSPs) Interface" on page 7-94.

The ADSP-21535 DSP does not support the MMSWS bit from previous SHARC DSPs.

DRAM Page Boundary Detection

Applications with large amounts of data may want to use DRAM memory for bulk storage. To simplify interfacing to page-mode or static-column DRAMs, the DSP detects page boundary crossings and outputs the PAGE signal to an external DRAM controller. Figure 7-5 shows an example DSP system with DRAM.

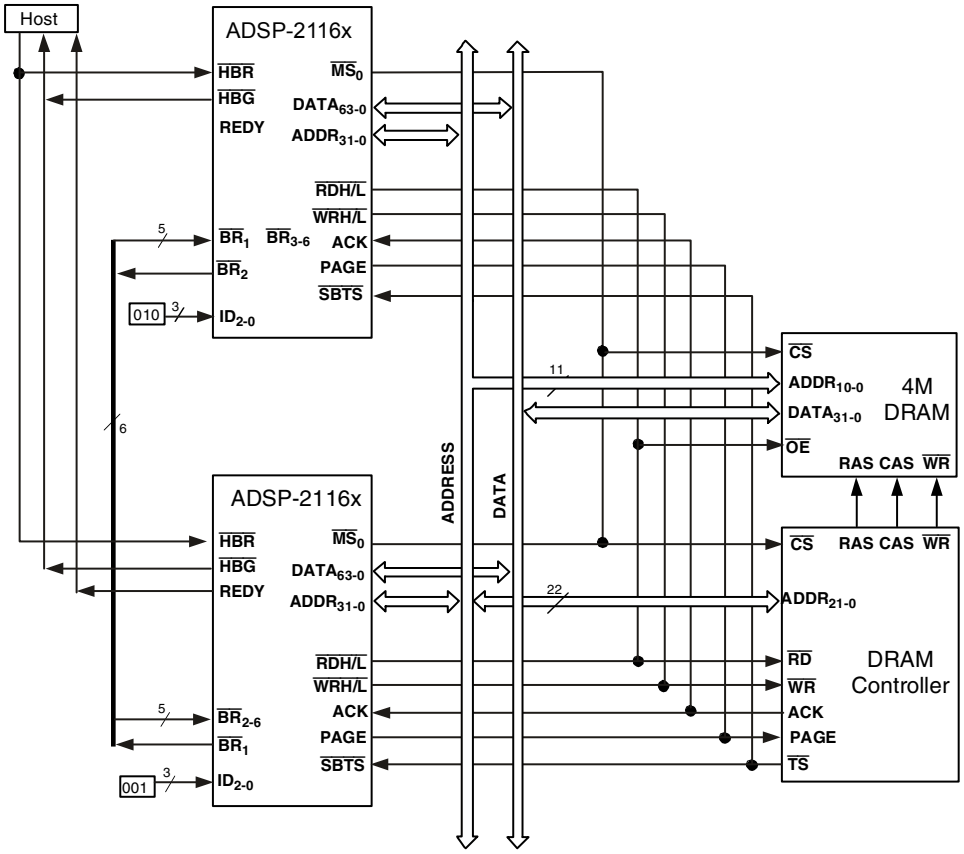


Figure 7-5. Example DRAM Interface

Different interfacing methods may be needed in some applications, especially if buffers are needed for the DRAM.

Page boundaries are user-defined. Boundaries must be programmed in the WAIT register. For more information, see "Setting Data Access Modes" on page 5-28. Automatic page boundary detection is provided by the DSP's PAGE signal. Systems may only place DRAM memory in bank 0 of external memory—the PAGE signal is only active within bank 0. Programs write the page size for page boundary detection in the PAGSZ field of the WAIT register.

The DSP asserts the PAGE pin when an external access crosses a page boundary and the address is within bank 0. The processor detects a boundary crossing by comparing each address output for bank 0 to the address of the last successful external access, which is stored in the I/O processor ELAST register. If a memory access is aborted—for example, due to a conditional write, the DSP does not assert the PAGE pin and does not update the current page in ELAST. Also, the DSP does not assert the PAGE pin or update the current page if the access is to multiprocessor memory space or to any memory space other than bank 0 of external memory space.

The PAGE pin remains asserted as long as the access is active. PAGE is not asserted if no access is performed. The current page is automatically invalidated and the PAGE pin asserted upon the next external access if: 1) the DSP loses mastership of the external bus to another DSP or to a host processor, or 2) the processor is reset. Programs should not read ELAST in the cycle immediately after it is written, because it may be in the process of updating.

The host bus request pin ($\overline{\text{HBR}}$) is disabled when the PAGE pin is asserted. Disabling $\overline{\text{HBR}}$ prevents the possibility of the DSP becoming a bus slave through deadlock resolution while the DRAM controller is servicing a page change.

In page DRAM systems, the DSP may need to recover from DRAM page fault conditions, using the Suspend Bus Three-State (SBTS) pin. External devices can assert the DSP's SBTS input to place the external bus address, data, selects, burst, and strobes in a high-impedance state. This input is sampled by the DSP on the rising edge of CLKIN. The DSP external bus outputs three-state later in the cycle in which SBTS is sampled asserted. If the DSP core attempts to access external memory while SBTS is asserted, the processor halts and the memory access does not complete until SBTS is sampled deasserted.



SBTS should only be used to recover from DRAM page faults or host processor/DSP deadlock condition. For more information, see "Deadlock Resolution" on page 7-87. In the case of DRAM page faults, SBTS allows the external DRAM controller to take control of the external bus. SBTS three-states the signals.

Table 7-2. Signals SBTS Three-States

ADDR31-0	RDH/L	BRST	DMAG1	MS3-0	CIF
PAGE	DATA63-0	WRH/L	DMAG2	BMS	CLKOUT

When the DSP uses SBTS for resolving bus deadlock, SBTS operates differently than when a host processor uses and SBTS and HBR. For more information see how the host processor uses SBTS and HBR as discussed at the end of the section "Synchronous Burst Read Transfers" on page 7-67.

When SBTS is asserted, the DSP places the external bus address, data, selects, and strobes in a high-impedance state for the following cycle. If an external access is underway when SBTS is asserted, the access is held off (as if ACK were deasserted), the bus is three-stated, and the memory access continues in the cycle after the deassertion of SBTS. If SBTS is asserted while no external access is occurring, the external bus pins are three-stated and the DSP continues running until it tries to perform an external access.

The DSP then halts. In this case, the memory access begins in the cycle after the deassertion of $\overline{\text{SBTS}}$.

When $\overline{\text{SBTS}}$ is deasserted, the DSP reasserts the $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, and $\overline{\text{DMAGx}}$ strobes—if they had been asserted prior to $\overline{\text{SBTS}}$—after the external address has become valid, asserting them at their normal timing within the cycle. The waitstate counter is reset.

$\overline{\text{SBTS}}$ differs from $\overline{\text{HBR}}$ in that $\overline{\text{SBTS}}$ takes effect in the next cycle, even if an external access is occurring but not finished. Systems should only use $\overline{\text{SBTS}}$ when the external access is to a device such as a DRAM or cache memory where the access must be held off in order to prepare for the access. Using $\overline{\text{SBTS}}$ at other times—such as during DSP-to-DSP accesses or when $\overline{\text{DMAGx}}$ is asserted—results in incorrect operation.

Timing External Memory Accesses

Memory access timing for external memory space and multiprocessor space is the same. For exact timing specifications, refer to the ADSP-21160 DSP Microcomputer Data Sheet.

The DSP can interface to external memories and memory-mapped peripherals that operate asynchronously with respect to CLKIN. The DSP also supports synchronous external memories and memory-mapped peripherals. Synchronous devices derive all of their bus timing with respect to CLKIN of the DSP.

The synchronous interface mode supports DMA burst transfers, which can significantly improve bus throughput for large, contiguous block transfers. The synchronous interface protocols are compatible with Synchronous Burst SRAMS (SBSRAMs) from a variety of vendors. In a multiprocessing system, the DSP must be the bus master in order to access external memory.

Asynchronous Mode Interface Timing

Figure 7-6 shows typical timing for an asynchronous read or write of external memory. Here, the CLKIN clock signal appears only to indicate that the access occurs within a single CLKIN cycle. All timing for the master DSP is derived synchronously from CLKIN. The asynchronous slave mode modifies the basic synchronous access to better support slaves whose timing is not derived from CLKIN.

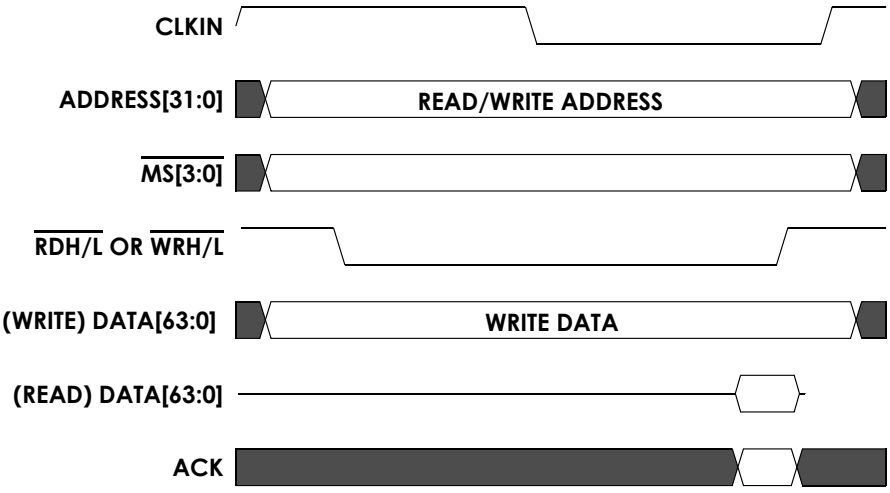


Figure 7-6. External Memory Asynchronous Access Cycle

Figure 7-7 shows timing relationships employed by the asynchronous external access mode.

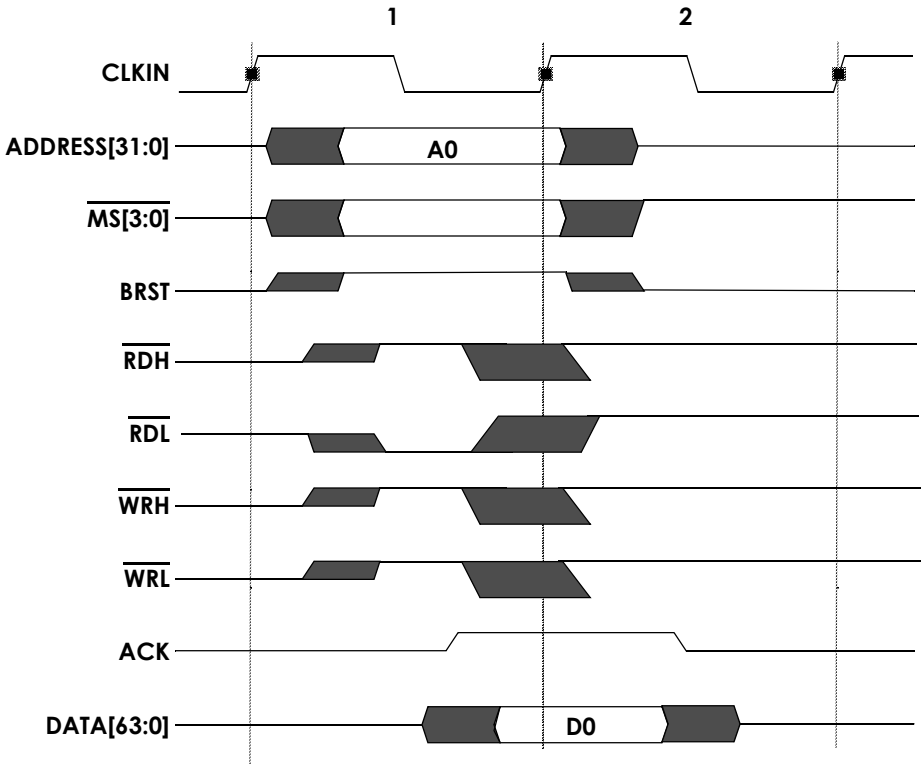


Figure 7-7. Asynchronous Access Timing Derivation

In this mode, the following occurs.

- The strobes assert and deassert based on timing derived from an internal clock whose frequency is twice that of the core clock. (This differs from synchronous mode where the strobes assert from the same edge.) The trailing edge timing is derived from the rising edge of the internal version of CLKIN.
- The $\overline{\text{MSx}}$ memory select lines are held stable for the entire access. (This differs from synchronous read or synchronous write—minimum 2-cycle—modes where the memory select lines are deasserted after the first ACK-ed cycle of the transfer.)
- For read operations, DATA63-32 are sampled by the DSP on the rising edge of the $\overline{\text{RDH}}$. DATA31-0 are sampled by the rising edge of $\overline{\text{RDL}}$. (This differs from synchronous mode where DATA63-0 are sampled by the internal version of CLKIN.)

Asynchronous Mode Read – Bus Master

DSP bus master reads of external memory, in asynchronous mode, occur with the following sequence of events as shown in Figure 7-6 on page 7-53.

1. The DSP samples ACK synchronously. If ACK is asserted, the DSP drives the read address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank. A memory select signal is not deasserted between successive accesses of the same memory bank. If ACK is sampled deasserted, the DSP waits one CLKIN cycle and samples ACK again.
2. The DSP asserts the read strobes. Strobe assertion is determined by the size and alignment of the data transfer. For more information on data alignment, see Figure 7-1 on page 7-2.

3. The DSP checks whether waitstates are needed. If so, the memory select and read strobe remain active for additional cycles. Waitstates are determined by a combination of the state of the external acknowledge signal (ACK) and the internally programmed waitstate count.
4. The DSP deasserts the read strobe(s) in the cycle where no further waitstates are indicated. The data bus (DATA63-0) is sampled on the rising edge of the read strobe(s).
5. If a Hold cycle is programmed for the accessed bank (via the EBxWS parameter of the WAIT register), the address bus and memory selects are held stable for an additional cycle. If initiating another read memory access to the same bank, the DSP drives the address and memory select for that access in the next cycle.

Asynchronous Mode Write – Bus Master

DSP bus master writes to external memory, in asynchronous mode, occur with the following sequence of events as shown in Figure 7-5 on page 7-15.

1. The DSP samples ACK synchronously. If ACK is asserted, the DSP drives the write address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank. A memory select signal is not deasserted between successive accesses of the same memory bank. The DSP also drives the write data (DATA63-0). If ACK is sampled deasserted, the DSP waits one CLKIN cycle and samples ACK again.
2. The DSP asserts the write strobes. Strobe assertion is determined by the size and alignment of the data transfer. For more information, Figure 7-1 on page 7-2.
3. The DSP checks whether waitstates are needed. If so, the memory select and write strobes remain active for additional cycles. Waitstates are determined by the state of the external acknowledge signal (ACK) and the internally programmed waitstate count.

4. The DSP deasserts the write strobes near the end of the cycle where no further waitstates are indicated.

5. The DSP three-states its data outputs, unless the next access is also a write to the same bank, or if a Hold cycle is programmed for the accessed bank using the EBxWS parameter of the WAIT register. If a Hold cycle is inserted, the address bus, data bus, and memory selects are held stable for an additional cycle. If initiating another memory access to the same bank, the DSP drives the address, memory select for the next access in the following cycle.

Synchronous Mode Interface Timing

Any slave addressed by a DSP in a bank configured for synchronous transfer mode must use a clock with the same frequency and phase characteristics to the clock which drives CLKIN on the DSP. The slave samples all inputs, and drives all outputs on the rising edge of this clock.

Except for zero-waitstate writes, the slave must assert ACK at least twice for each access; once to acknowledge the address/command (strobe assertion) and once (if not a burst) or more to acknowledge the data transfer.

The following notes apply to all synchronous access modes:

- A slave recognizes the start of a valid bus operation by synchronously sampling one or more of the strobes asserted and ACK asserted—but not by this slave, which would indicate the end of the transfer.

- For each of the non-burst, synchronous read/write accesses (except zero-waitstate writes), the master recognizes the end of the access as the cycle in which 1) the slave samples or drives data in response to a valid operation driven by the master (read or write), 2) the slave asserted ACK to the master {except for zero-waitstate write opera-

tions}, and 3) the number of waitstates for read or write access to that bank have occurred—asserting ACK does not terminate the wait count early.

- The program must select a number of waitstates that is consistent with the access time for the slave addressed by that external memory bank.
- For the zero-waitstate writes, the access can only be extended beyond one clock cycle by deasserting ACK in the cycle of the transfer. This extension can occur on back-to-back writes in which ACK is deasserted due to full write buffer capacity from the previous write, or slaves can asynchronously deassert ACK in the first cycle.
- Deasserting ACK during the initial command phase does inhibit waitstate count and change of bus signals. After the first ACK assertion, deasserting ACK for the data phase does not inhibit waitstate counting.
- Only one slave (or driver for ACK) should be allocated per external memory bank. More than one slave may introduce ACK drive contention.
- The read/write strobes for an access do not assert until ACK sampled asserted. This conditional strobe assertion delays the start of an access until ACK is asserted by the previous slave. This sampling is because the slave target of a single-cycle write operation may have to deassert ACK in the cycle after the bus cycle, to stall further writes to that slave. To provide a cycle for the previous slave to three-state its ACK driver before the next slave drives ACK, the next operation to a new bank must not launch on the bus.

- Write/read access stalls (no state change, other than internal wait-state counting) on the bus if ACK is deasserted in cycle(s) of data transfer.
- The last read/write operation must be ACK-ed before a transition to a new bus master (BTC), bank, or multiprocessor space slave occurs. The master always inserts an Idle cycle on this transition. No pipelining can occur across these boundaries.

Synchronous Mode Read – Bus Master

An example synchronous read cycle appears in Figure 7-8.

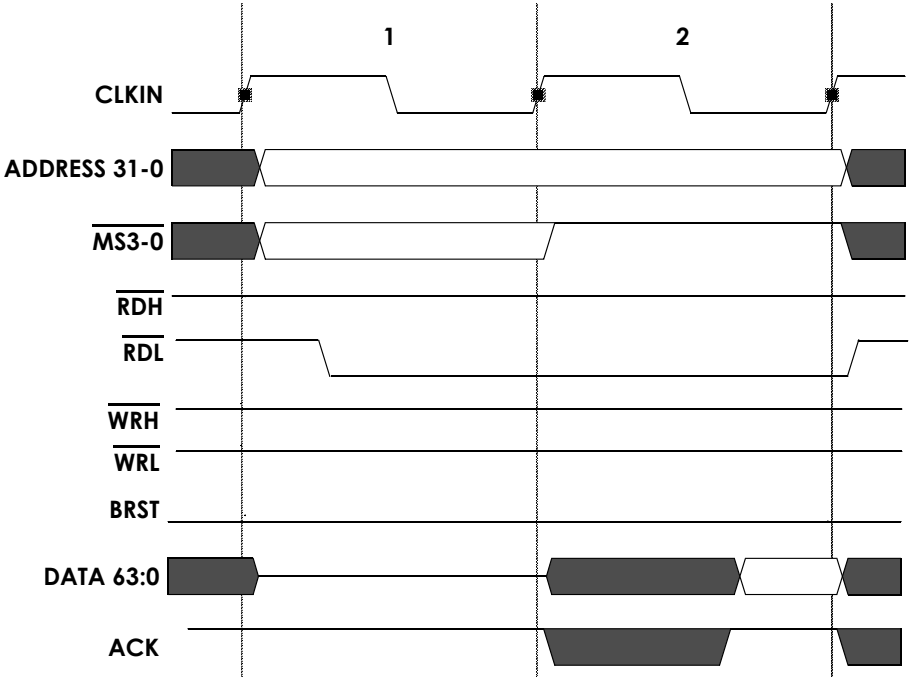


Figure 7-8. Typical Synchronous Read Timing

Propagation delays are not shown in this timing diagram. Because a synchronous access requires a rising clock edge for the slave to sample the asserted signals of the master (and for the master to sample slave), the minimum read access in the synchronous mode is two cycles.



In synchronous access mode, the waitstate selection in the WAIT register (EBxWS) must be 001 or greater. EBxWS=000 is not supported in synchronous access mode.

This example demonstrates a minimum latency, one-waitstate, 32-bit (normal word) read, from an even address in external memory (had the 32-bit access been to an odd 32-bit address, $\overline{\text{RDH}}$ would have asserted instead of $\overline{\text{RDL}}$.)



Slaves that do not support the entire 64-bit data bus width do not have to connect to both read strobes. Also, slaves that do not support bursting protocols do not need to connect to the BRST signal.

Bus master synchronous reads from external memory occur with the following sequence of events as shown in.

1. (cycle 1 in Figure 7-8 on page 7-25) If ACK is sampled as asserted at the beginning of cycle 1, the DSP drives the read address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank. The DSP asserts the $\overline{\text{RDH}}$/$\overline{\text{RDL}}$ strobes to indicate the size and alignment of the requested data. The read strobes are not deasserted between successive read accesses of the same memory bank. If the size or alignment changes, strobe assertion also changes. Strobe assertion is determined by the size and alignment of the data transfer. For more information on data alignment, see Figure 7-1 on page 7-2.
2. (cycle 2) If ACK was sampled as deasserted at the beginning of the cycle, the $\overline{\text{MSx}}$ strobes would remain asserted. If ACK was sampled asserted (as shown in Figure 7-8), the $\overline{\text{MSx}}$ strobes would deassert. The slave must be capable of detecting that $\overline{\text{MSx}}$ was asserted in

cycle 1 and retain this information internally. If ACK was deasserted by the previous slave (for a single-cycle write), deassertion of the $\overline{\text{MSx}}$ is delayed.

3. (cycle 2) The DSP checks whether more than one waitstates are needed. If so, the read strobes remain active for additional cycle(s). Waitstates are determined by a combination of the state of the external acknowledge signal (ACK) and the programmed waitstate count.
4. (end of cycle 2) The data bus (DATA63-0) is sampled on the rising edge of CLKIN.
5. (cycle 3) If initiating another read memory access to the same bank, the DSP drives the address, memory select, and strobes for the next access.

Figure 7-9 on page 7-28 shows back-to-back reads to the same bank with the second access stalled for one cycle by the slave deasserting ACK. This example assumes that the EBxWS=001 for this bank, indicating one internal waitstate.

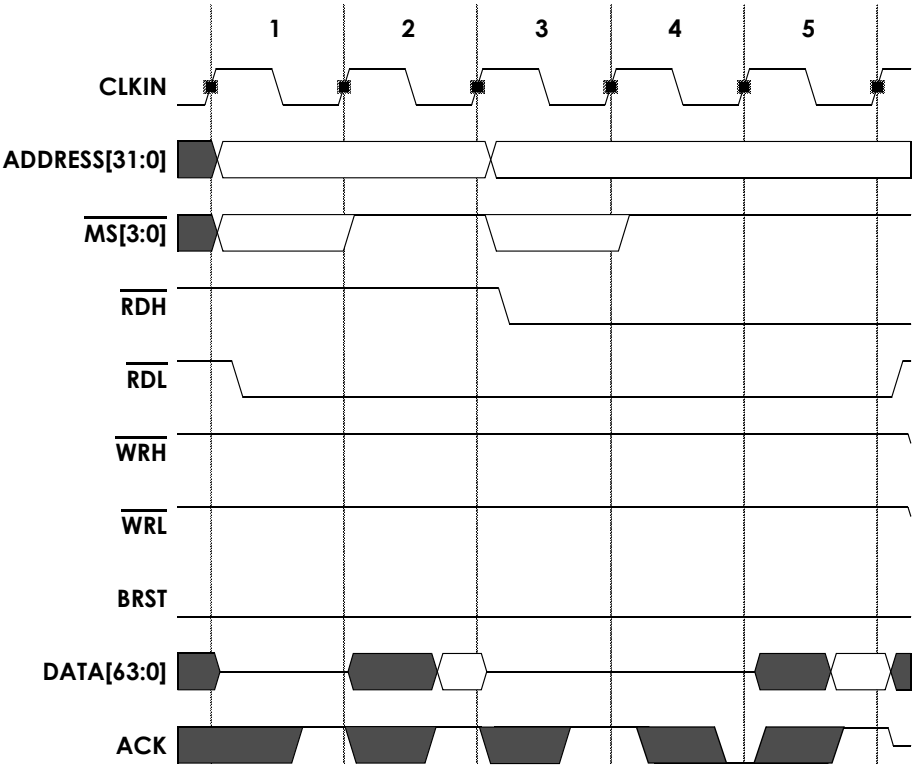


Figure 7-9. Two Synchronous Reads from Same Bank

Synchronous Write, Zero-Waitstate Mode

Figure 7-10 shows typical synchronous write cycle timing. Propagation delays are not shown in this timing diagram. Synchronous access requires a rising clock edge for the slave to sample the asserted signals of the master (and for master to sample slave). In the case of writes, the latency can be reduced to a single cycle if the slave always latches the bus signals on each clock cycle (it does not sample ACK). For example, the slave cannot sample

the bus, decode that it is being addressed as a slave, and sample the write data of the bus in the following cycle.

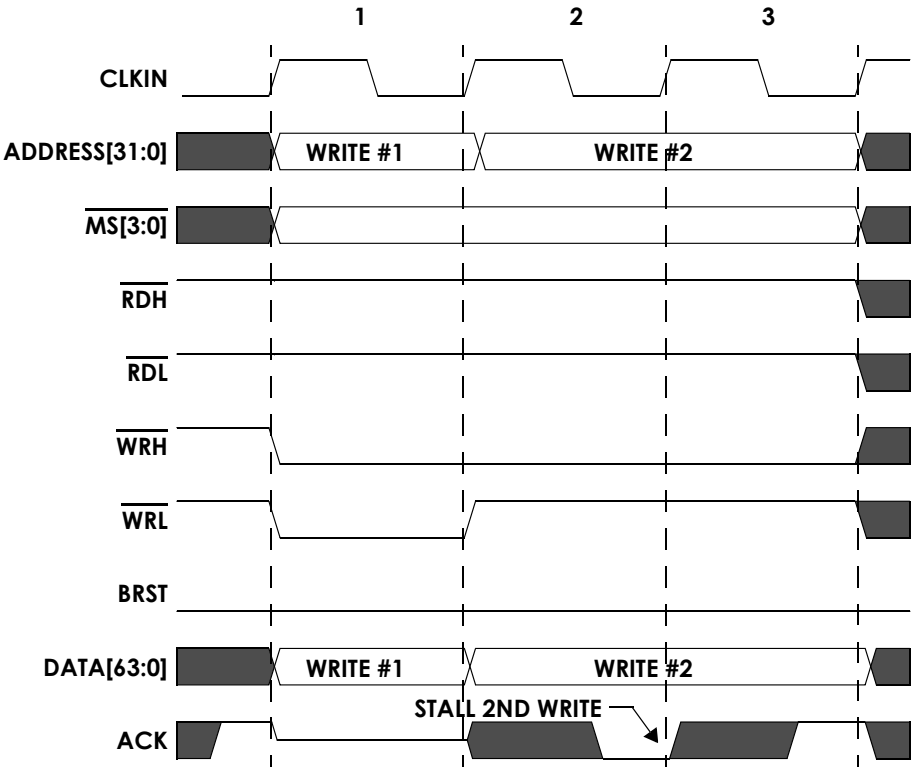


Figure 7-10. Typical Synchronous Write Example

The slave samples the bus each cycle and decodes the sampled value to determine if that slave was addressed by the write operation. If the slave's write queue goes full with that write, the slave deasserts ACK in the cycle after the write operation transferred on the bus. Any subsequent bus operation (read or write) stalls until ACK is sampled asserted, as shown in Figure 7-10.

The example demonstrates a minimum latency, zero-waitstate, 64-bit (Long word) write in cycle 1 followed by a write to the same bank that stalls because ACK is deasserted in cycle 2 in response to the write in cycle 1. The second access is a 32-bit write to an odd address in external memory. If the 32-bit access went to an even 32-bit address, $\overline{\text{WRL}}$ would have asserted instead of $\overline{\text{WRH}}$.

The zero-waitstate write mode provides the highest performance if the slave has sufficient write buffer storage. Systems should use this mode where the slave can always accept one write transfer (unless it has ACK deasserted) and can generally accept more than one write. If the slave has only one store buffer, such that it always deasserts ACK after the first write, the one-waitstate write mode may be the better choice. The zero-waitstate write mode is targeted towards ASIC/FPGA designs, which can likely implement multiple write buffers (including DSP as a slave), and fully pipelined synchronous devices such as SBSRAMs.



Slaves that do not support the entire 64-bit data bus width do not have to connect to both write strobes. Also, slaves that do not support bursting protocols do not need to connect to the BRST signal.

Bus master synchronous writes to external memory occur with the following sequence of events as shown in Figure 7-10 on page 7-29.

1. (cycle 1 in Figure 7-10) If ACK is sampled asserted at the start of cycle 1, the DSP bus master drives the write address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank. The DSP asserts the $\overline{\text{WRH}}$/$\overline{\text{WRL}}$ strobe(s) to indicate the size and alignment of the requested data. The write strobes are not deasserted between successive writes accesses of the same memory bank. If the size or alignment changes, strobe assertion also changes. Strobe

assertion is determined by the size and alignment of the data transfer. For more information on data alignment, see Figure 7-1 on page 7-2.

2. (cycle 1) The previous slave three-states ACK. The keeper latch on the DSP master keeps ACK at the asserted value until driven by the next slave. Note that the slave could have driven ACK through cycle 1. Only one slave is supported per bank, and any bank transition has an Idle cycle inserted to provide time for the slave to three-state ACK.

3. (cycle 2) The DSP is initiating another write memory access to the same bank. It drives the address, memory select, and strobes for the next access.

4. (cycle 2) The slave, having decoded that it received a valid write operation in the previous cycle, detects that it cannot accept further bus operations until the (or an element in the) write queue becomes available, so it deasserts ACK.

5. (cycle 3) The DSP samples ACK deasserted by the slave. It inserts waitstates until ACK is sampled asserted. The write ends in the cycle in which ACK is sampled asserted by the slave (end of cycle 3).

Figure 7-11 shows a zero waitstate write, followed by a synchronous read from the same bank. The slave addressed by both accesses determines in cycle 2 that it has no more write capacity. It deasserts ACK in this cycle, in response to the write in cycle 1. In cycle 3, the slave determines that it is now addressed by the master to perform a read and asserts ACK to acknowledge the transfer. The slave asserts ACK in cycle 4 when read data is available to complete the data transfer. The memory select for the read access is held asserted by the master until cycle 4, because ACK was deasserted in cycle 2. In this example, both operations use the full data bus

width, as indicated by both WRH/L and RDH/L strobes asserted in for the write and the read.

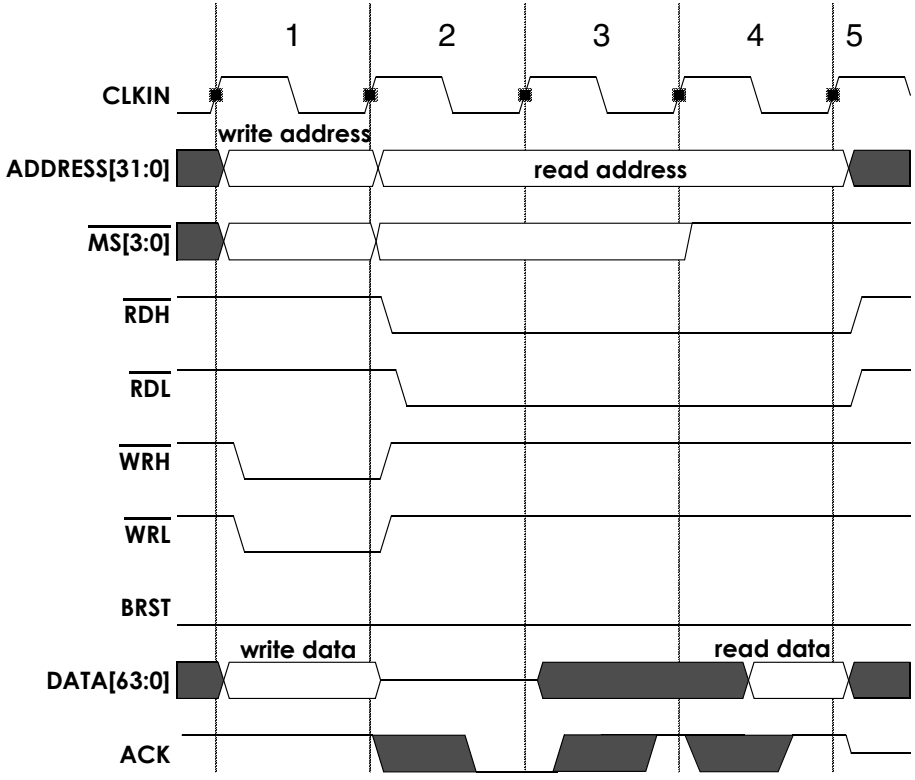


Figure 7-11. Synchronous Write Followed by Synchronous Read Example

Synchronous Write, One Waitstate Mode

Because some synchronous slaves cannot support a free-running latch function to capture zero-wait bus writes, the DSP also supports a minimum two-cycle (minimum one-waitstate) write access. This mode is set using the bank Access Mode bits (EBxAM). For more information on access modes, see Table A-19 on page A-49.

The one-waitstate, synchronous write access is shown in the second write of Figure 7-12.

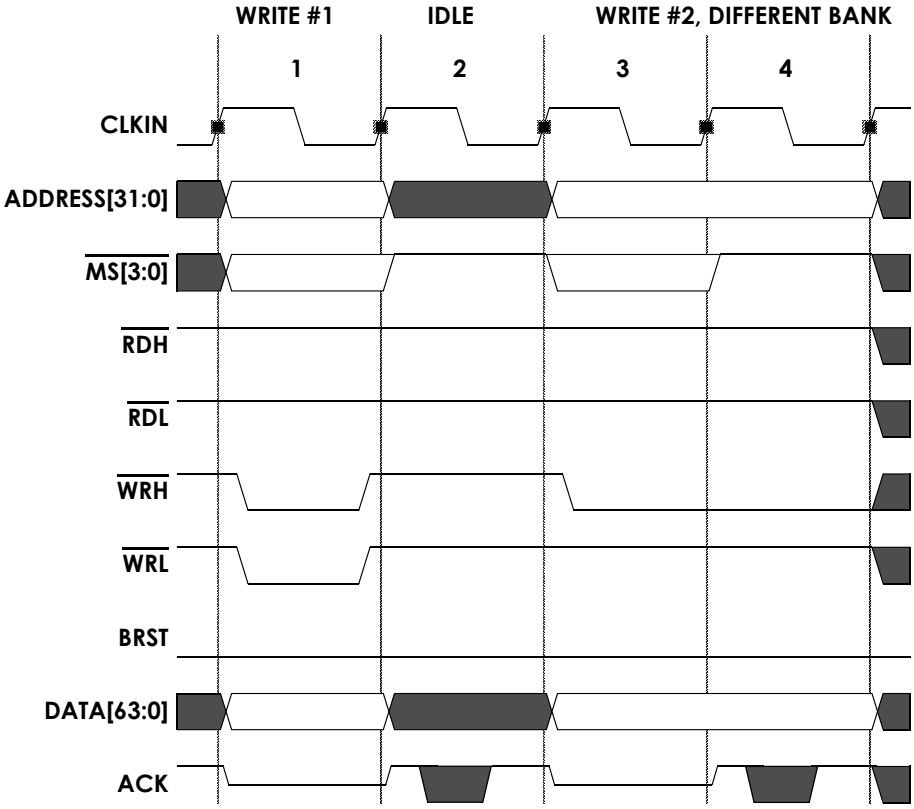


Figure 7-12. Asynchronous Write Followed by Synchronous Write - One-Waitstate Mode

In this example, the first access is to a bank configured for asynchronous writes (cycle 1). In Figure 7-12, this condition is shown by the deassertion of the write strobes before the rising edge of CLKIN for cycle 2. In cycle 2, a bank transition occurs, and an idle cycle is inserted to allow the slaves to transition ownership of ACK. In cycle 3, the second write begins, to a new

bank configured for one-waitstate write access. The address and data are held for a minimum of two cycles.

Similar to the synchronous read, $\overline{\text{MSx}}$ deassert in the next cycle (cycle 4), and the waitstate counter decrements if ACK is sampled asserted. The access can be held off the bus by deasserting ACK in cycle 2, or extended by deasserting ACK in cycle 3 (unlikely for a synchronous slave) or cycle 4.

Synchronous Burst Mode Interface Timing

Synchronous burst mode provides improved performance on synchronous operations, read operations in particular. The DSP supports a DMA-mastered (only) burst mode. If the addressed slave supports this burst transfer, after the one or more waitstates associated with access to the first 64-bit read data transfer, contiguous data can transfer on each subsequent clock cycle, up to a maximum of four 64-bit transfers. Burst accesses support only 64-bit data transfers. Partial data bus width transfers are not supported.

For burst transfers, the master drives the address of the first access on the bus during the entire burst transfer. The master does not increment the address for the slave. The maximum length of the burst transfer is four. So, slaves only need a 2-bit address incrementer to generate the offset address from the address driven by the master on the bus. Burst length determination as a function of initial address is shown in Table 7-3.

Table 7-3. Linear Burst Address Order

First Address[2:1] (external)	Second Address (internal)	Third Address (internal)	Fourth Address (internal)
00	01	10	11
01	10	11	Burst Terminated[1]
10	11	Burst Terminated[1]	
11	Burst Terminated[2]		

1 Master always terminates burst when internal address[2:1] = 11
2 Master transfers this case as a single synchronous access

If the DMA channel has sufficient data to transfer, it initiates a new burst transfer starting at ADDR2-1=00, 01, or 10 when it wins bus arbitration. Bursts always terminate when ADDR2-1=11.

An example of a synchronous burst read, of length three appears in Figure 7-13. Here, the bank employed in the transfer has 2 waitstates.

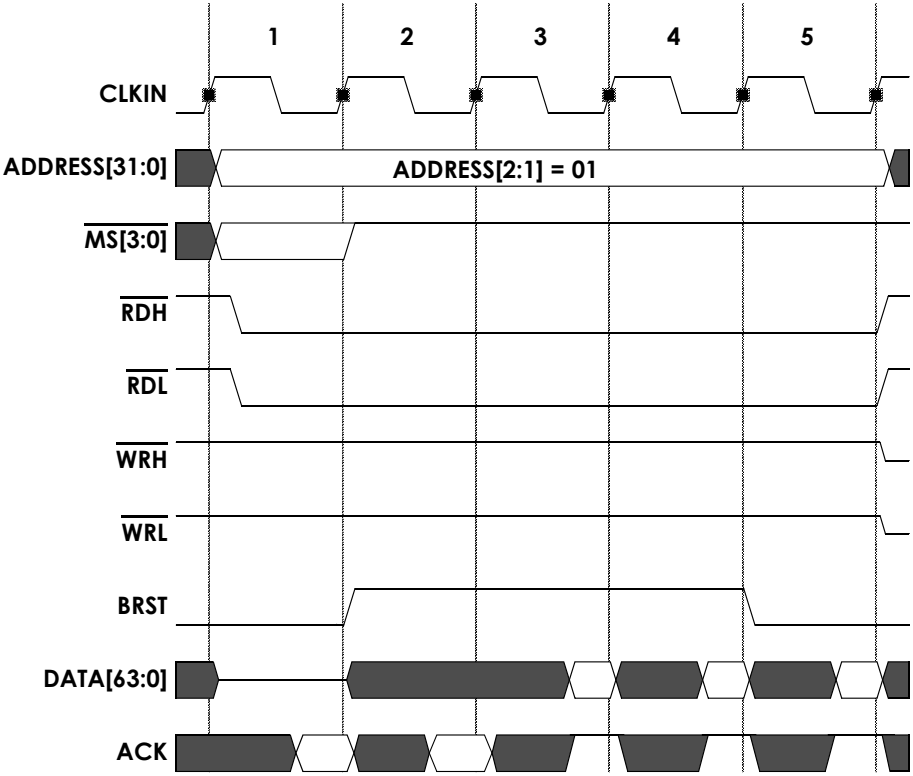


Figure 7-13. External Memory Synchronous Burst Read Example

Burst Length Determination

The DMA arbitration logic amortizes the initial access latency by bursting up to the maximum burst length of four when possible, assuming the channel is burst enabled. When a DMA channel wins internal I/O processor arbitration, the channel drives the internal buses as with a non-burst transfer. At the same time, the I/O processor detects whether it can perform a burst transfer, according to the following criteria.

1. The DMAC burst enable (MAXBL1-0) control bit field is set for that DMA channel. For more information on setting up a burst transfer, see the 64-bit External Burst Transfers discussion on page 6-26.
2. The EI register points to a 64-bit aligned address,
3. The EM register is set to 0 or 1. A value of 0 does not increment EI. This feature is useful when bursting to or from a registered data port, buffer, or register, such as the EPBx FIFOs of another DSP.
4. The EC register is >= 4 (four 32-bit words equals two 64-bit transfers).
5. The EPB FIFO for that channel has at least four 32-bit words to transfer for an external burst write or has at least four empty 32-bit elements to receive data for an external burst read.
6. The two least significant bits of the 64-bit DMA channel external address are not set (ADDR2-1 does not equal 11).

Burst Stall Criteria

If I/O processor determines that it can perform a burst transfer (according to the burst length criteria), the arbitration between the processor core

and I/O processor locks or parks the effective arbitration grant to that DMA channel until:

1. The DMA channel external ADDR2-1 = 11. By disconnecting the burst on this boundary, a modulo4 (ADDR31-1) is effectively implemented, which is required by SBSRAMs, and other slaves with limited address incrementing capability. For DSP-based systems, slaves only need a 2-bit counter to support the address incrementing function of the burst.
2. Space in the EPB FIFO drops to less than four 32-bit elements (if a external bus read), or less than four valid 32-bit elements for external bus writes. This almost full or empty detection is required by the master logic to deassert BRST on the cycle before the end of the burst.
3. EC goes to < 4; the burst pin must negate at EC=2.
4. $\overline{\text{HBR}}$ and $\overline{\text{SBTS}}$ are asserted on the external bus, indicating the deadlock resolution case in which the DSP must three-state its outputs and switch into slave mode. For more information, see "Deadlock Resolution" on page 7-87. Assertion of either signal alone does not terminate the burst early. $\overline{\text{HBR}}$ assertion does not receive an $\overline{\text{HBG}}$ until the burst finishes. $\overline{\text{SBTS}}$ assertion causes the master to three-state outputs and insert waitstates.

If any of these conditions occur, normal arbitration between the processor core and I/O processor for the external bus occurs. If the same bursting channel wins arbitration again, a new burst is initiated, introducing at least one lost or dead cycle in the burst throughput for reads.

When arbitration occurs, the DMA channel loses arbitration if any of the following conditions are detected:

1. Higher priority external request for the bus:
    a. $\overline{\text{HBR}}$ asserted.
    b. $\overline{\text{BRx}}$ asserted and BMAX time out has occurred.
    c. $\overline{\text{BRx}}$ asserted and $\overline{\text{PA}}$ asserted, but not by this master.
2. Higher priority internal I/O processor requester:
    a. Processor core request (DAGs or program sequencer)
    b. A higher priority request from another DMA channel or direct read/write access causes this channel to lose arbitration. For more information, see "I/O Processor"

Synchronous Burst Reads

External memory synchronous burst reads occur with the following sequence of events as shown in Figure 7-13 on page 7-35:

1. (cycle 1 in) If ACK is sampled asserted at the beginning of cycle 1, the DSP drives the read address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank.
2. (cycle 1) The DSP asserts both $\overline{\text{RDH}}$/$\overline{\text{RDL}}$ strobes to indicate a 64-bit read request of the slave.
3. (cycle 2) As with the non-burst synchronous read, the DSP deasserts the $\overline{\text{MSx}}$ output signal, asserts the BRST output signal and enables waitstate counting if ACK is sampled asserted at the end of cycle 1.
4. (cycle 2) The DSP checks whether more than one waitstates (2 waitstates for this example) are needed. If so, BRST and the read strobes remain active for additional cycle(s).

5. (cycle 3) The slave samples BRST asserted, informing it that the master requests at least one more 64-bit transfer after the current transfer is ACK-ed by the slave.
6. (cycle 3) The programmed number of waitstates (for example, 2) have been counted, and the slave is driving 64-bits of valid data and asserting the ACK signal. This ends the first access.
7. (cycle 4) The slave drives the next 64-bits of contiguous data and asserts ACK. If the slave needs more time to service any one transfer within the burst, it can deassert ACK to stall the bus transfer.
8. (cycle 4) The slave samples BRST asserted, informing it that the master requests at least one more 64-bit transfer.
9. (cycle 5) The master deasserts BRST to inform the slave that this is the last transfer of the burst. In this example, the master deasserts BRST due to the address modulo4 function. The two LSBs of the initial 64-bit address = 01. The slave increments the address as 01->10->11, the maximum offset it needs to support from the initial address.
10. (cycle 5) The slave drives valid data for the last transfer, and asserts ACK.
11. (cycle 6) If initiating another burst read memory access to the same bank, the DSP asserts the address, memory select, and strobes for the next access. This introduces at least two dead cycles in the back-to-back burst throughput, because the initial waitstate count applies to the first access of the second burst.
12. (cycle 6) With BRST sampled deasserted, the slave concludes its service of the burst request by three-stating the DATA63-0 and ACK drivers.

As a master, the DSP supports burst reads on each of the four external port DMA channels. Each channel has an independent burst enable con-

trol field (MAXBL1-0). For more information on setting up a burst transfer, see the 64-bit External Burst Transfers discussion on page 6-26.

As a slave, the DSP supports read bursts from internal memory or the EPBx buffers (with the EPBx read). For more information, see "Multiprocessor (DSPs) Interface" on page 7-94 and "Host Processor Interface" on page 7-51.



Because reads of the EPBx FIFO are destructive, the DSP slave must deassert ACK on each transfer of the burst to guarantee that it samples the deasserted BRST input before committing the EPBx FIFO read. If the system design employs a similar destructive read data buffer, similar precautions should be employed if burst reads of the buffer are supported.

Synchronous Burst Writes

The DSP can master burst read and write operations in the one-waitstate write access mode (EBxAM=10) if one or more DMA channels are configured appropriately. The DSP can master non-burst, zero-waitstate, writes every cycle. Burst write transfers are not supported in this access mode. Synchronous external devices which require at least one cycle of write access latency (for example, bus bridges, SDRAM controllers, and others) may be able to optimize throughput for burst write operations, based on the contiguous, incrementing block transfer information conveyed by the burst protocol. Burst accesses support only 64-bit data transfers. Partial data bus width transfers are not supported.

An example of a synchronous burst write appears in Figure 7-14. Here, the bank employed in the transfer has the 1 waitstate mode, for the first write of the burst.

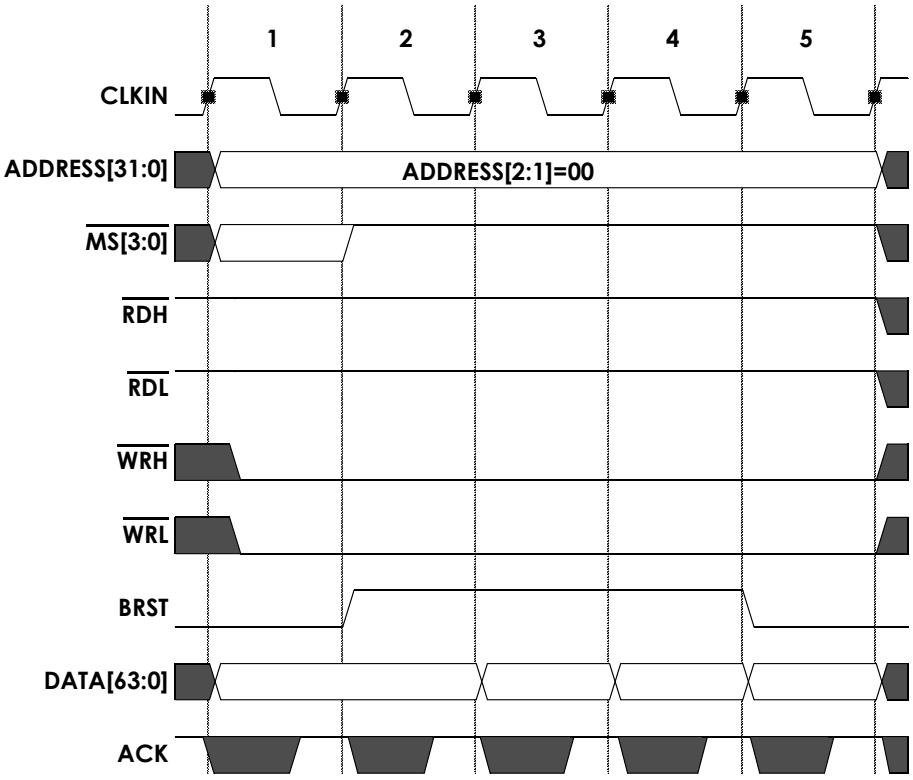


Figure 7-14. External Memory Synchronous Burst Write Example

External memory synchronous burst writes occur with the following sequence of events as shown in Figure 7-14 on page 7-41.

1. (cycle 1) If ACK is sampled asserted at the start of cycle 1, the DSP drives the write address and asserts a memory select signal ($\overline{\text{MS3-0}}$) to indicate the selected bank. The DSP also drives valid data in this cycle. The DSP asserts both $\overline{\text{WRH}}$/$\overline{\text{WRL}}$ strobes to indicate a 64-bit write command to the slave.
2. (cycle 2) The slave samples the write command and address. At this point, the slave does not see that a burst write is in progress—the access looks identical to a non-burst synchronous write. If the slave cannot accept the write command, it deasserts ACK in this cycle to stall the bus until it can. In this example, it has buffer capacity to accept all of the data of the burst, so ACK stays asserted.
3. (cycle 2) If ACK was sampled asserted at the start of the cycle, the DSP asserts the BRST output signal and deasserts the $\overline{\text{MSx}}$ output signal.
4. (cycle 3) The DSP samples ACK asserted by the slave at the start of the cycle, so it increments the data bus to the second of four data transfers within the burst.
5. (cycle 3) The slave samples BRST asserted at the start of the cycle, informing it that the master is writing at least one more 64-bit transfer. The slave samples the second of four data transfers within the burst and asserts ACK.
6. (cycle 4) The DSP samples ACK asserted by the slave at the start of the cycle, so it increments the data bus to the third of four data transfers within the burst.

7. (cycle 4) The slave samples BRST asserted at the start of the cycle, informing it that the master is writing at least one more 64-bit transfer. The slave also samples the third of four data transfers within the burst, and asserts ACK. If the slave needs more time to service any one transfer within the burst, it can deassert ACK to stall the bus transfer.

8. (cycle 5) The DSP samples ACK asserted by the slave at the start of the cycle, so it increments the data bus to the last of four data transfers within the burst. The master deasserts BRST to inform the slave that this is the last transfer of the burst.

9. (cycle 5) The slave samples BRST asserted at the start of the cycle, informing it that the master is writing at least one more 64-bit transfer. The slave samples the fourth of four data transfers within the burst and asserts ACK.

10. (cycle 6) If initiating another write burst memory access to the same bank, the DSP asserts the address, memory select, and strobes for the next access. This introduces at least one dead cycle in the back-to-back burst throughput, because the initial waitstate count applies to the first access of the second burst.

11. (cycle 6) With BRST sampled deasserted, the slave concludes its service of the burst request by three-stating the ACK driver.

As a master, the DSP supports burst writes on each of the four external port DMA channels. Each channel has an independent burst enable control field (MAXBL1-0). For more information on setting up a burst transfer, see the 64-bit External Burst Transfers discussion on page 6-26.



As a slave, ADSP-21535 DSP does not support burst writes. The DSP supports single cycle writes, so burst writes would provide no added performance improvement.

Using External SBSRAM

The DSP can connect to a variety of synchronous burst static RAMs (SBSRAMs) with a glueless interface—no external logic required. These synchronous memories can provide high throughput, especially when employing the burst read transfer modes. The DSP has features to support SBSRAMs from a number of memory vendors.

The DSP can support flow-through, pipelined and ZBT SBSRAMs. Where bus frequency and system organization features like trace lengths, capacitive loading, and termination characteristics allow, using flow-through devices delivers lower latency and higher system performance.

The DSP can support SBSRAMs on any of the four external memory banks. The DSP supports SBSRAM single transfer reads and writes and SBSRAM burst read transfer operations.


Single cycle burst write transfers are not supported.

SBSRAM support is enabled by configuring the bank access mode (EBxAM) bits for synchronous, 1-cycle writes and waitstate (EBxWS) bits for 1 wait-state (flow-through SBSRAMs) or 2 waitstates (fully pipelined SBRAMs). For more information on programming access modes and waitstates, see the WAIT register bits in Table A-19 on page A-49.

If burst read transfer capability is needed, one or more of the external port DMA channels must be configured appropriately. For more information on setting up a burst transfer, see the 64-bit External Burst Transfers discussion on on page 6-26. Because burst transfers are controlled at the DMA channel, the DMA sequence must make sure that the DMA burst transfer addresses a memory bank or slave that supports the read burst transfer.

Figure 7-15 and Table 7-4 on page 7-46 show how the DSP I/O should be connected to the SBSRAM I/O. Table 7-4 assumes a 512KByte

SBSRAM array consisting of one bank of two 3.3V, 32K x 32 devices. The names of the SBSRAM signals may vary from one vendor to another.

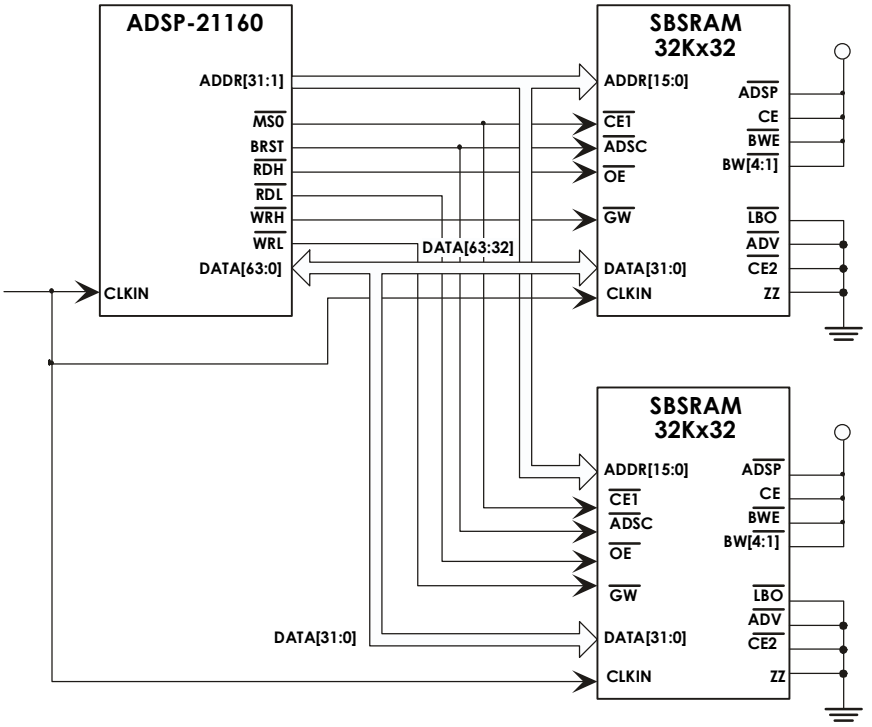


Figure 7-15. SBSRAM System Interface Example

Figure 7-15 is for illustrative purposes—actual system designs may differ and must be carefully analyzed to determine the actual system topology.

Table 7-4. ADSP-21535 to SBSRAM Signal Mapping

DSP	SBSRAM	Comment
CLKIN	CLK	Both devices driven by same input clock
ADDR16-1	ADDR15-0	Read/Write strobes decode bit 0 of address
$\overline{\text{MSx}}$	$\overline{\text{CE}}$	Chip Enable, active low
BRST	$\overline{\text{ADSC}}$	Address Status Controller, active low
$\overline{\text{RDH}}$	$\overline{\text{OE}}$	Asynchronous Output Enable of SBSRAM #1, active low
$\overline{\text{RDL}}$	$\overline{\text{OE}}$	Asynchronous Output Enable of SBSRAM #2, active low
$\overline{\text{WRH}}$	$\overline{\text{GW}}$	Global Write Enable of SBSRAM #1, active low
$\overline{\text{WRL}}$	$\overline{\text{GW}}$	Global Write Enable of SBSRAM #2, active low
DATA63-32	DATA31-0	I/O of SBSRAM #1 (High word of bus, odd address)
DATA31-0	DATA31-0	I/O of SBSRAM #2 (Low word of bus, even address)
No connect	CE	Chip Enable, active high, Always Asserted (Vdd)
No connect	$\overline{\text{CE2}}$	Second Chip Enable, Always Asserted (GND)
No connect	$\overline{\text{ADSP}}$	Always Deasserted (Vdd)
No connect	$\overline{\text{ADV}}$	Always Asserted (GND)
No connect	$\overline{\text{BWE}}$	Byte Write Enable, Always Deasserted (Vdd)
No connect	$\overline{\text{BW}}$4-1	Byte Write Selects, Always Deasserted (Vdd)
No connect	$\overline{\text{LBO}}$	Linear Burst Order, active low, always asserted (GND)
No connect	ZZ	Sleep Mode Enable, active high, always deasserted (GND)

The SBSRAM devices are fully synchronous devices, except for the output enable. The DSP issues commands and updates the SBSRAM address latches, as a controller, using the $\overline{\text{ADSC}}$ input of the SBSRAMs, rather than the $\overline{\text{ADSP}}$ processor input. Using the $\overline{\text{ADSC}}$ SBSRAM input enables single cycle writes and simplifies SBSRAM deselect operations.

By always asserting the $\overline{\text{ADV}}$ (advance address) input to the SBSRAM, the device is always attempting to burst. This input is a do not care when $\overline{\text{ADSC}}$ is asserted. Because the BRST/$\overline{\text{ADSC}}$ signal is always low for a single access or

the first access of a burst, the SBSRAM always updates its address latches correctly. For the subsequent transfers (up to three, after the initial access) of a read burst, the SBSRAM samples BRST/$\overline{\text{ADSC}}$ high. The asserted $\overline{\text{ADV}}$ correctly advances the internal address count of the SBSRAM.

Figure 7-16 on page 7-47 demonstrates a burst read of the flow-through SBSRAM.

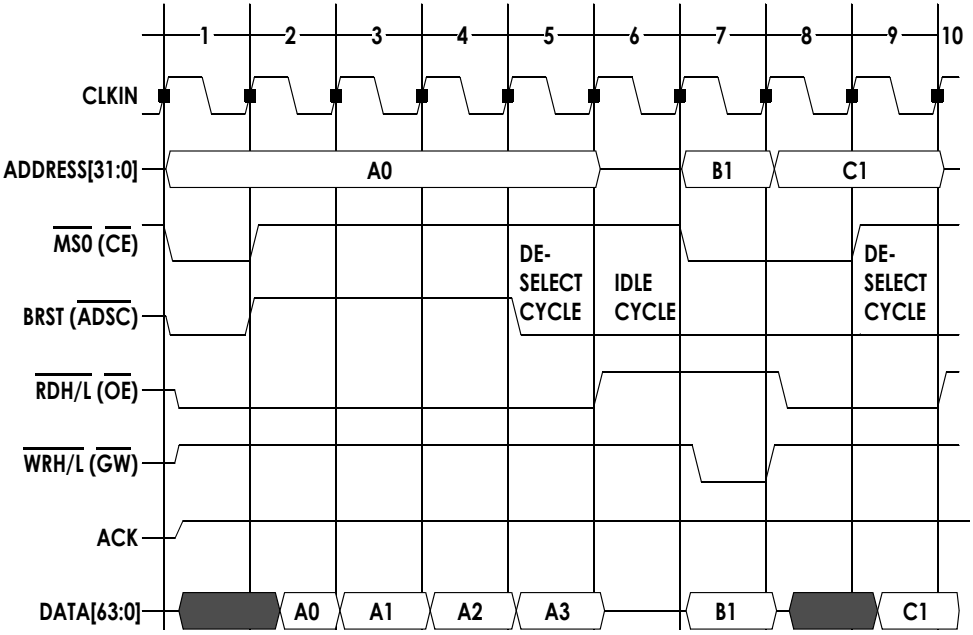


Figure 7-16. SBSRAM – Burst Read, Single Write, Single Read

The DSP issues four types of bus operations to the SBSRAMs, as shown in Table 7-5.

Table 7-5. SBSRAM Partial Truth Table

SBSRAM Operation	$\overline{CEI}$ $\overline{MSx}$	$\overline{ADSC}$ BRST	$\overline{ADV}$[1]	$\overline{GW}$ $\overline{WRx}$	$\overline{OE}$ $\overline{RDx}$	I/O
Read cycle, begin burst	L[2]	L	X	H	L	Data
Write cycle, begin burst	L	L	X	L	H	Hi-Z
Read cycle, continue burst	X	H	L	H	L	Data
Deselect Cycle	H	L	X	X	X	Hi-Z
All other signal inputs held static per Figure 7-7						

1 $\overline{ADV}$ statically held asserted, low
2 L=low, H=High, X=don't care, Hi-Z=three-stated, high impedance output

Single read or write transfers, and the first transfer of a burst read, employ the read or write cycle, begin burst bus operation. Burst write transfers are not supported. The subsequent transfers (up to three) of a read burst employ the read cycle, continue burst bus operation. The last cycle of any read access performs a deselect bus operation to make sure that the SBSRAM data buffers remain three-stated for accesses to other banks.

The write operations are achieved by configuring the appropriate bank of DSP to synchronous minimum one-cycle write mode. The synchronous

read waitstate count should be programmed to one for flow-through SBSRAMs, or two for fully pipelined SBSRAMs.



The DSP's page detection function is not needed for SRAM memory systems.



SBSRAMs are not stalled, or suspended, by assertion of ACK in this configuration. Systems should not deassert ACK during any SBSRAM access. The DSP has a weak pull-up device on ACK; ACK does not need to be driven during an access to a slave which does not or cannot control ACK.

The read is followed by a single write to the SBSRAM, which is followed by a single read of the SBSRAM. For burst operations, the deasserting BRST is not required in the last cycle of the burst transfer. The DSP's burst protocols also support ASIC/FPGA systems in which the pipelined end-of-burst indicator may be of value.

It is possible to increase the SBSRAM array size from the example. This increase can come from using higher density devices or implementing multiple banks of SBSRAM. Multiple banks are possible using the depth expansion capability of the SBSRAMs and the multiple memory select outputs of the DSP.

Executing Instructions From External Memory

For systems that execute instructions from external memory, the system must include a bank of 48-bit or 64-bit wide memory that is allocated specifically to program memory. This dedicated bank for instructions is required because fetch addresses from the program sequencer are pointers to 48-bit locations—the DSP does not translate the fetch address into a 32-bit address for external memory. The system can select the instruction

bank using one of the memory select ($\overline{\text{MS3-0}}$) pins or the Core Instruction Fetch ($\overline{\text{CIF}}$) pin.



DSP performance is reduced significantly when executing instructions directly from external memory.

For the instruction bank, the system can use $\overline{\text{CIF}}$ as a separate bank select. $\overline{\text{CIF}}$ has the same timing as the $\overline{\text{MS3-0}}$ outputs, but $\overline{\text{CIF}}$ asserts only for an instruction fetch from external memory (depends on fetch from sequencer, not address in memory). If the instruction fetch occurs to the address range of one of the external banks, the DSP also asserts the memory select for that bank.



The ADSP-21535 DSP supports the Core Instruction Fetch ($\overline{\text{CIF}}$) pin for executing instructions from external memory. This pin is not available on the previous SHARC DSPs.

To fetch instructions from external memory, the system uses either of the following methods:

Connect a dedicated bank of 48-bit wide memory to DATA63-16 pins and use $\overline{\text{CIF}}$ or $\overline{\text{MS3-0}}$ as the memory selects. The DSP uses the full address bus (ADDR31-0 including LSB of the address) to address this bank. A bank of 64-bit wide memory also can be used this way.

- Connect a bank of 64-bit wide memory, store the 48-bit instructions MSB aligned in this bank, and use external address translation to generate the appropriate address on an instruction fetch. The system can use $\overline{\text{CIF}}$ as an indication of the fetch.



Using either of the above methods, the DSP asserts both $\overline{\text{RDH/L}}$ for instruction fetch accesses.



The address translation in the second method is required to accommodate the unpacked instruction locations. The Program Sequencer issues sequential addresses for each fetch, but each unpacked instruction word uses two (32-bit) memory locations.

The DSP only asserts $\overline{\texttt{CIF}}$ during instruction fetches from external memory. Other types of external memory accesses (such as 40-bit data accesses or DMA transfers of packed instructions) do not use $\overline{\texttt{CIF}}$. For more information on 40-bit data accesses in external memory, see "Internal Data Bus Exchange" on page 5-7.

For more information on packed data transfers from external memory, see the packed data discussion on page 6-27.

Host Processor Interface

The DSP's host interface supports connecting the DSP to 16- or 32-bit microprocessor buses. By providing an address, a data bus, and memory control signals—such as read, write and chip select—a host may access any device on the DSP bus as if it were a memory. Figure 7-17 shows an example of how to connect a host processor to the DSP.

The DSP accommodates either synchronous or asynchronous data transfers, allowing the host to use a different clock frequency. Transfers at speeds up to the full CLKIN clock rate are supported.



Figure 7-17 shows all lines needed for an asynchronous host interface. Systems using a synchronous interface do not need $\overline{\texttt{CS}}$ or REDY, and the address comparator might not be needed, depending on the host processor's requirements.

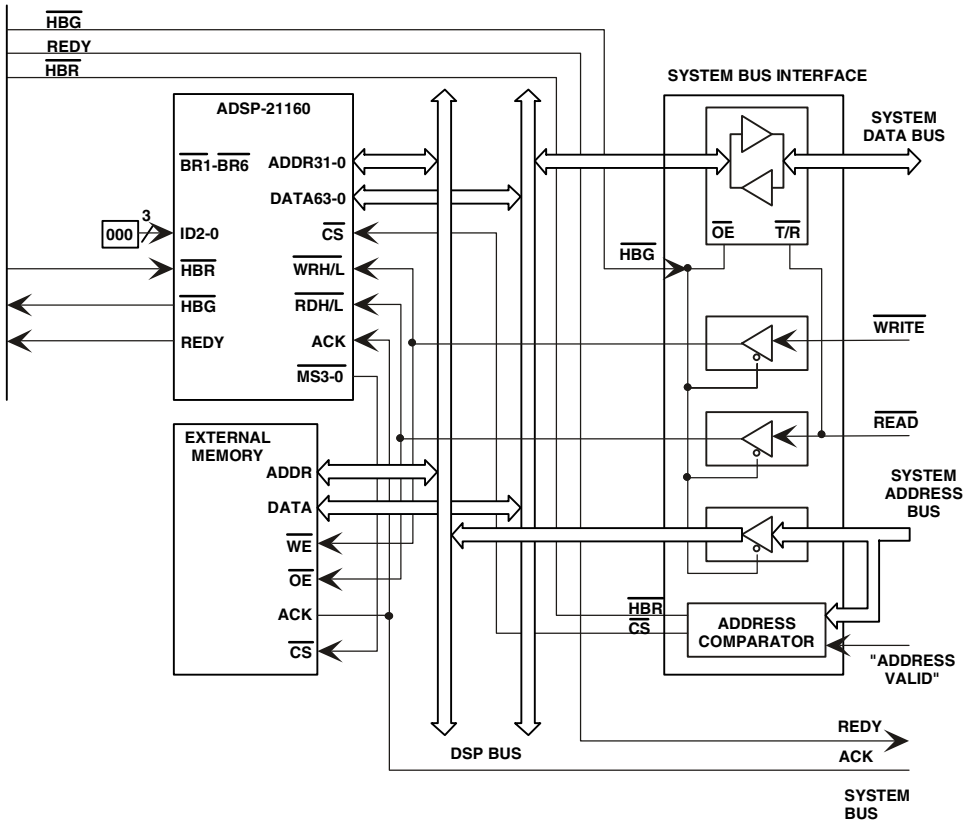


Figure 7-17. Example DSP-to-Host System Interface

Table 7-6 defines the DSP pins used in host processor interfacing.

Table 7-6. Host Interface Signals

Signal	Type	Definition
$\overline{\text{HBR}}$	I/A	Host Bus Request. The host processor must assert $\overline{\text{HBR}}$ to request control of the DSP's external bus. When $\overline{\text{HBR}}$ is asserted in a multiprocessing system, the DSP that is bus master relinquishes the bus and asserts $\overline{\text{HBG}}$. To relinquish the bus, the DSP places the address, data, select, and strobe lines in a high-impedance state. $\overline{\text{HBR}}$ has priority over all DSP bus requests ($\overline{\text{BR1-6}}$) in a multiprocessing system.
$\overline{\text{HBG}}$	I/O	Host Bus Grant. $\overline{\text{HBG}}$ acknowledges an $\overline{\text{HBR}}$ bus request, indicating that the host processor may take control of the external bus. $\overline{\text{HBG}}$ is asserted (held low) by the DSP until $\overline{\text{HBR}}$ is released. In a multiprocessing system, $\overline{\text{HBG}}$ is output by the DSP bus master and is monitored by all others.
$\overline{\text{CS}}$	I/A	Chip Select. The host processor asserts $\overline{\text{CS}}$ to select the DSP (for asynchronous transfer protocol).
REDY	O (o/d)	Host Bus Acknowledge. The DSP deasserts REDY (low) to add waitstates to an asynchronous access of its internal memory or IOP registers by a host. This pin is open-drain output (o/d) by default, but can be programmed with the ADREDY bit of SYSCON register to be active drive (a/d). REDY is only output if the $\overline{\text{CS}}$ and $\overline{\text{HBR}}$ inputs are asserted.
$\overline{\text{SBTS}}$	I/S	Suspend Bus Three-state. External devices can assert $\overline{\text{SBTS}}$ (low) to place the external bus address, data, selects, and strobes in a high-impedance state for the following cycle. If the DSP attempts to access external memory while $\overline{\text{SBTS}}$ is asserted, the processor halts and the memory access does not complete until $\overline{\text{SBTS}}$ is deasserted. $\overline{\text{SBTS}}$ should only be used to recover from PAGE faults or host processor/DSP deadlock.
I=Input, S=Synchronous, (o/d)=Open Drain, O=Output, A=Asynchronous, (a/d)=Active Drive		

The host accesses the DSP through the DSP's external port. Figure 6-8 on page 6-68 shows a block diagram of the external port, I/O processor, and FIFO data buffers, illustrating the on-chip data paths for host-driven transfers. The four external port DMA channels are available for use by

the host—DMA transfers of code and data can be performed with low software overhead.



The ADSP-21535 DSP supports the host interface protocols of the previous SHARC DSPs. Also, the ADSP-21535 DSP provides new synchronous interface protocols that support the 64-bit data bus and burst transfers of sequential data.

The host processor requests and controls the DSP's external bus with the host bus request ($\overline{\text{HBR}}$) and host bus grant ($\overline{\text{HBG}}$) signals. Host logic does not need to duplicate the distributed multiprocessor arbitration protocol of the DSPs.

After the host gets control of the DSP bus, the host may transfer data either synchronously or asynchronously. The host bus may be 16, 32, or 64 bits wide for synchronous transfers, but only 16 or 32 bits wide for asynchronous transfers.

For asynchronous transfers, the host also uses the chip select ($\overline{\text{CS}}$) and ready (REDY) signals. After getting control of the bus, the host can directly read and write the internal memory of the DSP. The host can also read and write to any of the DSP's I/O processor registers, including the EPBx FIFO buffers. The host uses certain I/O processor registers to control and monitor the DSP (such as SYSCON and SYSTAT) and to set up DMA transfers. DMA transfers are controlled by the DSP's I/O processor after they are set up by the host. In a multiprocessor system, the host can access the internal memory and I/O processor registers of every DSP.

Data written to and read from the DSP can be packed or unpacked into different word widths. When the width of the host bus is 16 or 32 bits, the DSP can pack data into 32 or 48-bit words. The DSP attempts to gather two 32-bit words into one single 64-bit internal transfer where possible. When the width of the host bus is 64 bits (synchronous transfer modes only), the DSP can pack 48-bit instructions so four instructions are transferred in three 64-bit transfers (maximum throughput), or the DSP handles unpacked data so only 48-bits of the 64-bit transfer are treated as

valid data. The host packing mode control bits (HPM) in the SYSCON register configure data packing and unpacking.

Acquiring the Bus

For a host processor to gain access to the DSP, the host must first assert $\overline{\text{HBR}}$, the host bus request signal. $\overline{\text{HBR}}$ has priority over all $\overline{\text{BRx}}$ multiprocessor bus requests. When asserted, $\overline{\text{HBR}}$ causes the current DSP master to give up the bus to the host after the DSP finishes the current bus operation. If the current operation is a burst transfer, the change in bus mastership interrupts the transfer on a modulo4 boundary.

The current DSP bus master signals that it is transferring ownership of the bus by asserting $\overline{\text{HBG}}$ (low) when the current bus operation ends. The cycle in which control of the bus is transferred to the host is called a Host Transition Cycle (HTC).

Figure 7-7 on page 7-20 shows the timing for the host acquiring the bus. $\overline{\text{HBG}}$ is asserted while the bus master releases control of the bus and remains asserted until $\overline{\text{HBR}}$ is sampled deasserted by the DSP. The cycles in which control of the bus is released by the bus master is called the DSP's Bus Transition Cycle (BTC). $\overline{\text{HBG}}$ freezes DSP multiprocessor bus arbitration during the time that the host owns the bus. $\overline{\text{HBG}}$ may be used to enable the host's signal buffers, as shown in Figure 7-17 on page 7-52, Figure 7-23 on page 7-85, and Figure 7-24 on page 7-86. While $\overline{\text{HBG}}$ is asserted in a multiprocessor system, the DSPs continue to assert their $\overline{\text{BRx}}$ outputs, as in normal operation, but no BTCs occur. The current DSP bus master keeps

its $\overline{BRx}$ output asserted throughout the entire time the host controls the bus.

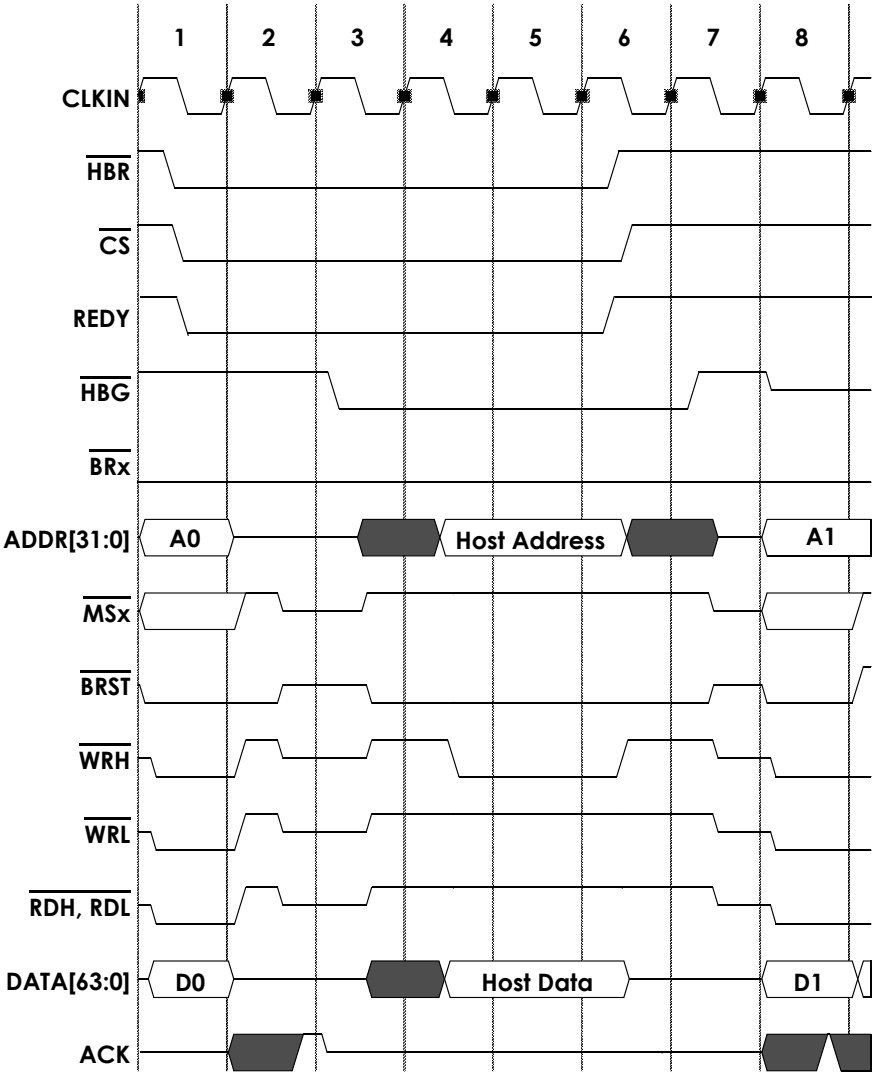


Figure 7-18. Example Timing for Host Acquisition of Bus

After the host gets control of the bus, the host can choose to perform synchronous or asynchronous transfers with the DSP or other system components. To initiate asynchronous transfers, the host asserts (low) the $\overline{\text{CS}}$ and $\overline{\text{HBR}}$ inputs of the DSP that it intends to access and performs the read or write. The DSP does not respond to $\overline{\text{CS}}$ until $\overline{\text{HBG}}$ is asserted. To initiate synchronous transfers, the host keeps all DSP $\overline{\text{CS}}$ pins deasserted (high) and reads or writes to the DSPs' multiprocessor memory space.

The host may also communicate directly with system peripherals, such as SBSRAMs. These transfers occur using the protocol of the peripheral or using the external handshake mode of DMA channels 11 and 12 to control the memory or peripheral. With DMA handshaking, the host only needs to source or sink the data with the correct timing. Either of these solutions may require additional hardware support for the host.

The host is responsible for driving the following signals during the HTC in which it gains control of the bus: ADDR31-0, $\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, $\overline{\text{WRL}}$, $\overline{\text{DMAGx}}$ (if employed in the system), and PAGE (if the PAGE function is employed in the system application). These signals must be driven by the host while the host is bus master. Also, the host must drive or weakly pull up or down the $\overline{\text{MS3-0}}$, BRST, CLKOUT, $\overline{\text{DMAG1}}$, and $\overline{\text{DMAG2}}$ signals as required. The DSP bus master three-states these lines, letting the host use them.

The DSP with device ID=000 or 001 enables internal pull-up devices on the $\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, $\overline{\text{RDL}}$, $\overline{\text{RDH}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, $\overline{\text{DMAR1}}$, $\overline{\text{DMAR2}}$, $\overline{\text{DMAG1}}$, and $\overline{\text{DMAG2}}$ signals. The pull-up provides a weak current source to hold these signals in the deasserted state when driven to that state.



Excessive system noise can cause these weakly driven signals ($\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, $\overline{\text{RDL}}$, $\overline{\text{RDH}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$, $\overline{\text{DMAR1}}$, $\overline{\text{DMAR2}}$, $\overline{\text{DMAG1}}$, and $\overline{\text{DMAG2}}$) to be sampled asserted.

The DSP with device ID=000 or 001 enables its keeper latches on ADDR31-0 and DATA63-0, BRST, PAGE, and CLKOUT, so these signals are weakly pulled to the last value driven on them if any of these signals remain undriven for multiple cycles.

During read-modify-write operations, the host should keep $\overline{\text{HBR}}$ asserted to avoid temporary loss of bus mastership. $\overline{\text{HBR}}$ must remain asserted until after the host completes the last data transfer.

The following restrictions apply to bus acquisition by the host.

- If $\overline{\text{HBR}}$ is asserted while the DSP is in reset, the DSP does not respond with $\overline{\text{HBG}}$ until after reset and multiprocessor synchronization is completed.
- The host should keep $\overline{\text{HBR}}$ asserted until after the host completes its last data transfer and is ready to give up bus ownership.
- If $\overline{\text{SBTS}}$ is asserted after $\overline{\text{HBR}}$, the DSP enters slave mode and suspends any unfinished access to the external bus.
- In uniprocessor systems (with ID2-0=000), the host must assert $\overline{\text{CS}}$ in the same cycle as $\overline{\text{HBR}}$ to initiate an asynchronous access.
- If the host is to execute both synchronous and asynchronous accesses during a single bus grant, it must allow at least one cycle to pass after the last access before switching $\overline{\text{CS}}$.
- Synchronous accesses may not be used in systems with only one DSP (with ID2-0=000).
- Synchronous burst writes to DSPs are not supported. Synchronous burst reads of DSP internal memory and the I/O processor's EPBx FIFOs are supported.

After the host finishes its task, it can relinquish control of the bus by deasserting $\overline{\text{HBR}}$. The DSP bus master responds by deasserting $\overline{\text{HBG}}$ in the cycle after sampling $\overline{\text{HBR}}$ deasserted. In the cycle following deassertion of $\overline{\text{HBG}}$, the DSP bus master assumes control of the bus and normal multiprocessor arbitration resumes.

Asynchronous Transfers

To initiate asynchronous transfers after acquiring control of the DSP's external bus, the host must assert the $\overline{\texttt{CS}}$ input of the DSP to be accessed. The host then drives the address of the memory location or I/O processor register to access. To simplify the hardware requirements for external interface logic, only the address bits shown in Table 7-7 need to be driven.

Table 7-7. Address Fields For Asynchronous Host Accesses

Address Bits[1]	Comments
ADDR7-0	Must be driven in all cases
ADDR16-8	Must be driven only if the S field indicates an internal memory access
ADDR19-17	S field[2] – Must be driven "000" for IOP register accesses, "01m" for internal memory normal word accesses, or"1nn" for short word accesses
and either	
ADDR22-20	M field[2] – Must be driven "000" to deselect other DSPs, if present,
or	
ADDR31-23	E field[2] – One of the lines 31–23 driven as "1"

1 Setup and hold times for these address lines are specified in the DSP data sheet.
2 For a complete description of these address fields, see "ADSP-21160 DSP Memory Map" on page 5-12.

Table 7-7 applies to all asynchronous host access cases, including multiprocessor systems. Fewer address bits may need to be driven depending on the system. For example in a uniprocessor DSP system, the host need not drive the ADDR22-20 address pins.

Host direct reads and writes are possible for Normal word or Short word data. Host access to Long word data on DSP slaves is not supported.

Normal words are accessed if the ADDR19-17 address pins are 01m, where m is the most significant bit of the Normal word address. Short words are

accessed if the ADDR19-17 address pins are 1nn, where nn are the two most significant bits of the Short word address.

When using asynchronous transfers and direct access to internal memory is not required, only the lower 8 bits, ADDR7-0, need be supplied by the host. The upper address bits can be configured as explained in Table 7-7.



The ADSP-21535 DSP does not support the Instruction Word Transfer (IWT) function from previous SHARC DSPs. 48-bit instructions can be transferred by configuring the host packing mode to one of the 48-bit internal transfer modes. For direct write operations, the correct size information is tracked through the FIFO.

Asynchronous write operations are latched at the I/O pads in a four-deep FIFO buffer; this buffer is called the slave write FIFO and appears in Figure 6-8 on page 6-68. This buffering allows previously written words to be re-synchronized while a new word is being written and allows asynchronous writes to occur at up to the full CLKIN clock rate of the DSP.

A host may write to several DSPs simultaneously (a broadcast write), by asserting each of their $\overline{\text{CS}}$ pins. Each DSP accepts the write as if it were the only device being addressed. Because the REDY output is wire-OR'ed (if configured as an open-drain output), REDY only appears asserted when all selected DSPs are ready, unless REDY is actively pulled up. ACK is not active when $\overline{\text{CS}}$ is asserted.

To eliminate the need for a host to drive the multiprocessor address lines (ADDR22-20) in systems with only one DSP (ID2-0=000), such DSP does not recognize synchronous accesses to these addresses. The host must drive these address lines if the DSP's ID2-0 is anything other than 000. To account for buffer delays when sampling the REDY signal, systems must make sure that REDY is properly re-synchronized by the host.

Asynchronous Transfer Timing

When a DSP's $\overline{\text{CS}}$ chip select is asserted (low), the selected DSP deasserts the REDY signal. Refer to the *ADSP-21160 DSP Microcomputer Data Sheet* for exact timing specifications.

As shown in Figure 7-19, the DSP deasserts REDY in response to $\overline{\text{CS}}$. The host can assert $\overline{\text{CS}}$ before or after $\overline{\text{HBR}}$ is asserted, but the DSP does not reassert REDY until after $\overline{\text{HBG}}$ is asserted and a $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ strobe is applied. This condition is true only if a $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ strobe is active when $\overline{\text{HBG}}$ is asserted. Otherwise, this timing is determined by the t_{TRDYHG} switching characteristic specified in the "*Multiprocessor Bus Request and Host Bus Request*" timing data in the ADSP-21160 DSP Microcomputer Data Sheet.

REDY is asserted prior to the $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ being asserted and becomes deasserted only if the DSP is not ready for the read or write to complete—the only exception is when $\overline{\text{CS}}$ is first asserted. The REDY pin is an open-drain output to facilitate interfacing to common buses. It can be changed to an active-drive output by setting the ADREDY bit in the SYSCON register.



The ADSP-21535 DSP's asynchronous transfer timing is similar to that of previous SHARC DSPs with one important difference. The DSP has two read strobes ($\overline{\text{RDH}}$ and $\overline{\text{RDL}}$) and two write strobes ($\overline{\text{WRH}}$ and $\overline{\text{WRL}}$). Each of these strobes enables or masks 32-bits of the 64-bit data bus. Only $\overline{\text{RDH}}$ and $\overline{\text{WRH}}$ are employed on asynchronous transfers using the host packing mode support for 16/32-bit transfers. See "External Port Buffer Modes" on page 6-17.

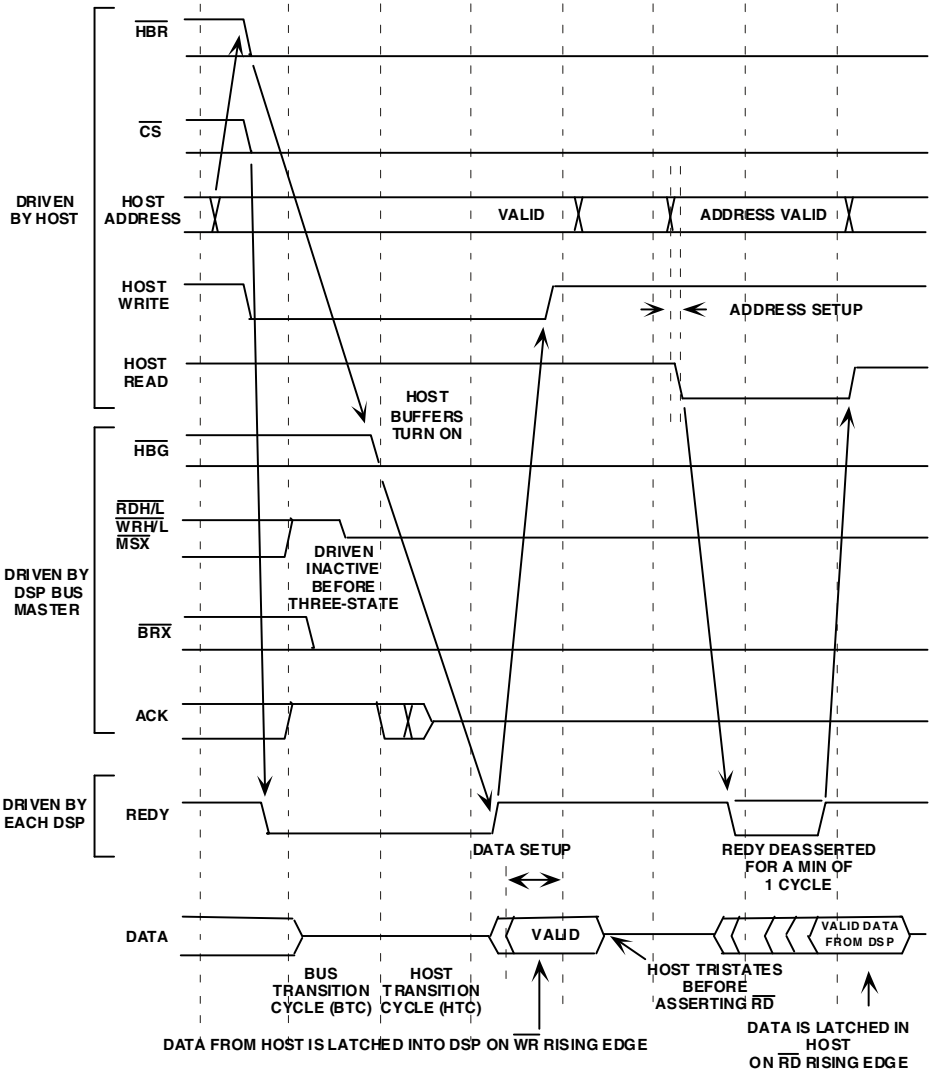


Figure 7-19. Example Timing For Host Read and Write Cycles

Figure 7-19 shows the timing of a host write cycle, including details of data packing and unpacking. This timing applies to the example host

interface hardware shown in Figure 7-24 on page 7-86 and has the following sequence.

1. The host asserts the address. $\overline{\text{HBR}}$ and $\overline{\text{CS}}$ are decoded from the host bus interface address comparator and do not need to be supplied directly by the host. The selected DSP deasserts REDY immediately.
2. The host asserts $\overline{\text{WRH/L}}$ and drives data according to the timing requirements specified in the ADSP-21160 DSP Microcomputer Data Sheet.
3. The selected DSP asserts REDY when it is ready to accept the data. This transition occurs after the current bus master has completed its current transfer and has asserted $\overline{\text{HBG}}$. $\overline{\text{HBG}}$ enables the host interface buffers to drive onto the DSP bus.
4. The host deasserts $\overline{\text{WRH/L}}$ when REDY is high and stops driving data.
5. The selected DSP latches data on the rising edge of $\overline{\text{WRH/L}}$.

After the first word, the write sequence is:

1. The host asserts $\overline{\text{WRH/L}}$ and drives data according to the timing requirements specified in the ADSP-21160 DSP Microcomputer Data Sheet.
2. The DSP deasserts REDY if it is not ready to accept data.
3. The host deasserts $\overline{\text{WRH/L}}$ when REDY is high and stops driving data.
4. The selected DSP latches data on the rising edge of $\overline{\text{WRH/L}}$.

More than one DSP may have its $\overline{\text{CS}}$ pin asserted at any one time during a write, but not during a read because of bus conflicts.

Figure 7-19 on page 7-62 also shows the timing of a host read cycle. This timing applies to the bus interface hardware in Figure 7-24 on page 7-86 and has the following sequence:

1. The host asserts the address. $\overline{\text{HBR}}$ and the appropriate $\overline{\text{CS}}$ line are decoded by the host bus interface address comparator. The selected DSP deasserts REDY immediately and asserts $\overline{\text{HBG}}$.
2. The host asserts $\overline{\text{RDH/L}}$.
3. The selected DSP drives data onto the bus and asserts REDY when the data is available.
4. The host latches the data and deasserts $\overline{\text{RDH/L}}$.

After the first word, the read sequence is:

1. The host asserts $\overline{\text{RDH/L}}$.
2. The selected DSP deasserts $\overline{\text{REDY}}$ then asserts REDY, driving data when it becomes available.
3. The host deasserts $\overline{\text{RDH/L}}$ when REDY is high and latches the data.

Synchronous Transfers

Synchronous transfers are defined by both master and slave deriving bus timing (sampling bus inputs and driving bus outputs) from the same clock input (same clock frequency and phase). Synchronous transfers potentially offer significantly higher throughput than asynchronous transfers, but may require additional synchronization logic.

To perform synchronous transfers, the $\overline{\text{CS}}$ input is not asserted and the host must act like another DSP in a multiprocessor system. The host must generate an address in the multiprocessor memory space of a DSP, assert the $\overline{\text{RDH/L}}$ or $\overline{\text{WRH/L}}$ strobes (and BRST if a burst read transfer) and sink or source data.

For examples of synchronous transfers, see "External Memory Interface" on page 7-3 and "Multiprocessor (DSPs) Interface" on page 7-94.

Synchronous accesses are not supported in uniprocessor systems in which the ID2-0=000. Synchronous accesses are possible in uniprocessor systems in which the ID2-0=001 or in multiprocessor systems. To perform synchronous accesses in a multiprocessor system, the host must drive the address pins ADDR22-20 with a value of 1-7 to select one of the DSPs (by its ID2-0) or one of the ADDR31-23 address pins must be driven high to select an address in external memory. For more information on using these address pins, see Table 7-7 on page 7-59.

For synchronous host transfers, the DSP uses its ACK signal—instead of REDY—to add waitstates to an access. The host must wait for the DSP to assert ACK. Synchronous accesses are not recognized during the Host Transition Cycle. This prevents any spurious access from occurring while the external host buffers are starting to drive the address, data, and strobes.



ACK may glitch during the HTC and should not be relied on until the following cycle.

When a DSP is responding to a synchronous read access, it drives valid data only in the cycle in which it asserts ACK. The DSP three-states the data bus in the cycles following assertion of ACK in response to a synchronous host read, even if the host continues to assert $\overline{\text{RDH}}/\overline{\text{RDL}}$.

Synchronous Broadcast Writes

The timing in Figure 7-20 on page 7-66 demonstrates two synchronous broadcast writes from the host. Broadcast writes address multiple slaves. Master and slaves employ a different protocol for ACK-acknowledgment of the broadcast write, relative to other bus operations. For broadcast writes, slaves employ a wired-OR protocol in which they drive ACK deasserted (only if required) in alternate cycles, starting with the cycle after the write is driven on the bus. The master pre-charges ACK high in alternate cycles, starting with the second cycle after the broadcast write.

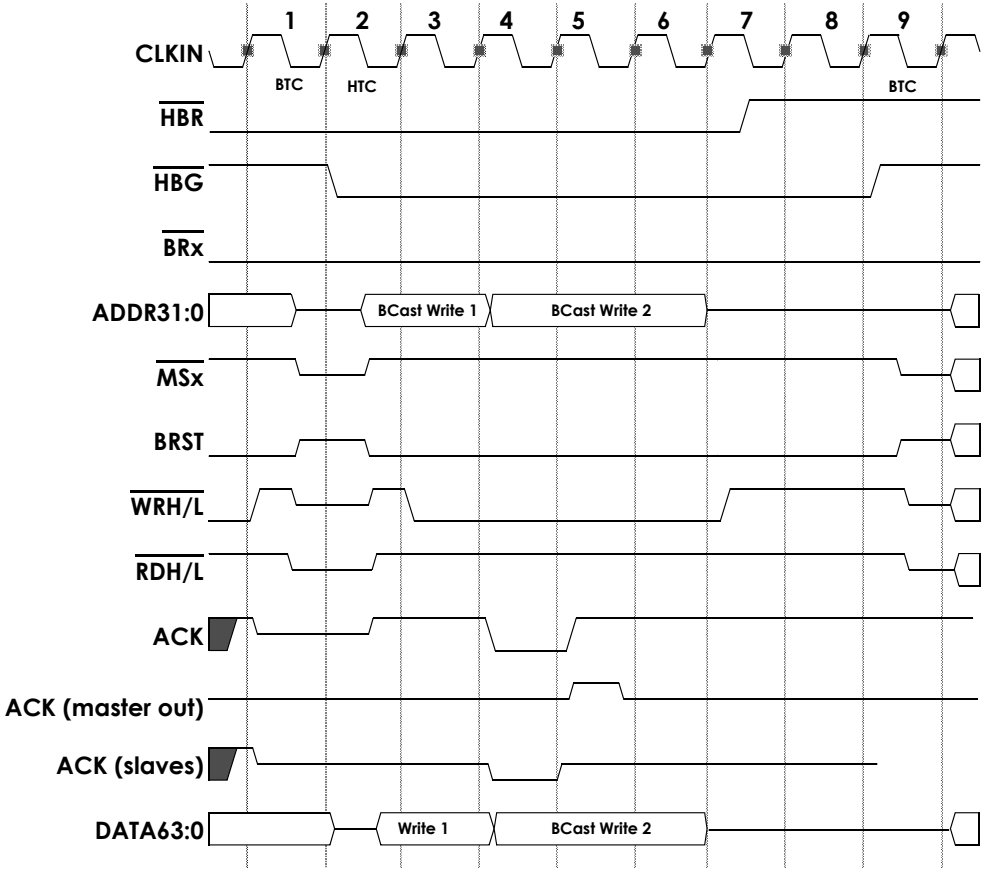


Figure 7-20. Synchronous Broadcast Write Example

In this example, the first broadcast write is accepted by all slaves at the end of cycle 3. One or more slaves must stall any further access until one has capacity to accept the next write operation. This stall is accomplished by the slaves deasserting ACK in cycle 4. The master must pre-charge ACK in cycle 5, the second cycle after the first broadcast write. All slaves can accept more operations by cycle 6, and none of the slaves drive ACK deasserted again.

The host samples ACK asserted in cycle 6 and completes the second broadcast write as shown by deasserting the write strobes in cycle 7.

Even though the host deasserted $\overline{\text{HBR}}$ in cycle 7, the DSP does not respond by deasserting $\overline{\text{HBG}}$ until cycle 9.

Synchronous Burst Read Transfers

As a slave, the DSP supports synchronous burst read transfers from the EPBx FIFOs or direct reads from internal memory. Burst write transfers are not supported, because single-cycle, non-burst writes can provide as much write bandwidth as burst writes. Burst reads are supported as contiguous, aligned, 64-bit data transfers up to a maximum length of four 64-bit transfers. The DSP slave increments the address if the burst read access is from internal memory. The slave address increment function is only supported for ADDR2-1. The host cannot burst across a modulo4 (ADDR2-0) address boundary as shown in Table 7-3 on page 7-34.

To perform a burst read transfer from an EPBx buffer, the host issues a starting burst address pointing to one of the EPBx buffers. The DSP slave does not increment an EPBx burst read address. The modulo4 (ADDR2-0) address boundary restriction does not apply in this case.

Long burst transfers tie up the external bus and may prevent or significantly degrade system response to potentially higher priority events. The DSP has no way of truncating or disconnecting a host access.

If $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ are asserted while an external DMA access is occurring, $\overline{\text{HBG}}$ is not asserted until the access is completed.

The DSP also supports burst transfers, which can be truncated by assertion of $\overline{\text{HBR}}$ and $\overline{\text{SBTS}}$. If the DMA transaction was a burst transfer, when the host relinquishes control of the local bus, the DSP resumes the burst transfer, starting at the address of the last operation that did not complete.

Slave Direct Reads and Writes

The host can directly access the internal memory and I/O processor registers of a DSP by reading or writing the appropriate address in multiprocessor memory —known as a direct read or direct write access. Each DSP bus slave monitors addresses driven on the external bus and responds to any that fall within its region of multiprocessor memory.

These accesses are invisible to the slave DSP's processor core. They do not degrade internal memory or internal bus performance. Direct access is important, because it lets the processor core continue program execution uninterrupted.

The host can directly read or write the I/O processor registers to control and configure the DSP or to set up DMA transfers.

IOP Shadow Registers

To ease host and multiprocessor system operations, the DSP I/O processor registers include registers that shadow or mirror some processor core system registers, including the program counter (`PC`), and `MODE2` registers. These registers facilitate system start up and debug, by letting the host (or another DSP in an multiprocessor system) interrogate these processor core registers. These shadow registers are read only and lag the value of the registers they shadow by one internal core clock. For more information, see "PC Shadow Register (PC_SHDW)" on page A-53 and "MODE2 Shadow Register (MODE2_SHDW)" on page A-53.



The silicon revision field of the `MODE2` shadow register `MODE2_SHDW` is now used for differentiating between silicon revisions. These corresponding bits in the `MODE2` (foreground) register are now reserved. The application program must read the `MODE2_SHDW` regis-

ter bits [31:25] to identify the silicon revision. MODE2_SHDW is a memory-mapped IOP register whose address is 0x11. See Figure 7-21 on page 7-69.

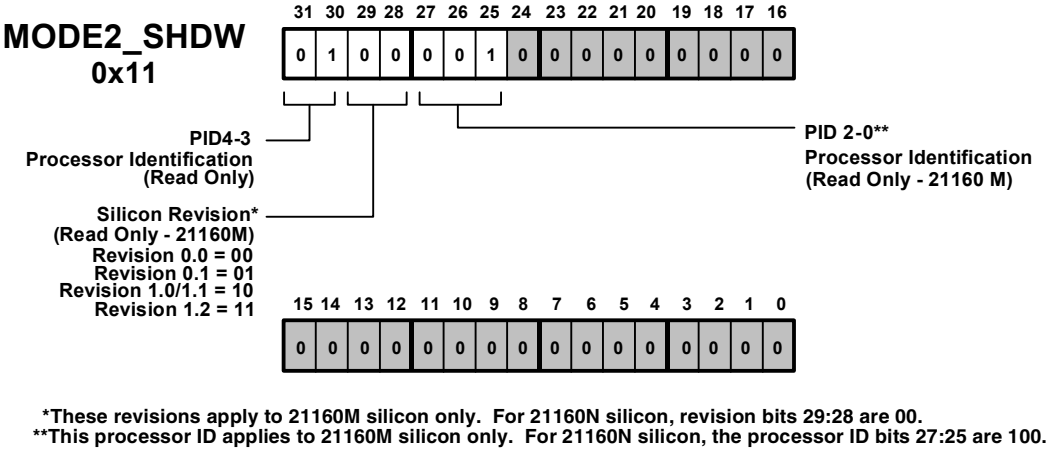

MODE2_SHDW
0x11
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
PID4-3
Processor Identification
(Read Only)
Silicon Revision*
(Read Only - 21160M)
Revision 0.0 = 00
Revision 0.1 = 01
Revision 1.0/1.1 = 10
Revision 1.2 = 11
PID 2-0**
Processor Identification
(Read Only - 21160 M)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*These revisions apply to 21160M silicon only. For 21160N silicon, revision bits 29:28 are 00.
**This processor ID applies to 21160M silicon only. For 21160N silicon, the processor ID bits 27:25 are 100.

Figure 7-21. Mode2 Shadow Register

Instruction Transfers

For 16- or 32-bit host interfaces, the DSP can pack and unpack 48-bit instruction or 40-bit Extended Precision Normal word data based on the host packing mode selected with the HPM bits in the SYSCON register. Host processors can achieve maximum throughput by transferring packed instructions to or from internal memory, using the full 64-bit data bus width synchronously. For more information, see the **Packed** versus **Unpacked** instruction discussion on on page 6-26.

In packed 64-bit direct reads or writes of 48-bit memory, the program must translate the addresses from an instruction word (Iword) pointer to a Normal word (Nword) pointer as follows.

1. Nword address= [(Iword address - Block base) * 3/2] + Block base
2. If the host transfers a group of instructions to the DSP, the instructions go into DSP memory starting at an internal memory address (for example, 0x40100). In internal memory, the instructions are packed instructions—48-bit instructions 48-bit aligned—not padded to align to 64-bit boundaries.
3. Programs can find the Normal word starting address of the host transfer to the DSP by scaling the least significant 16-bits of the internal 48-bit instruction address by 3/2. For example, the Normal word address that corresponds to instruction address 0x40100 is:

```
Nword address= [(0x40100 - 0x40000) * 3/2] + 0x40000
   = 0x40180
```



The ADSP-21535 DSP does not support the IWT bit from previous SHARC DSPs.

Host Direct Writes and Reads

The DSP supports synchronous and asynchronous direct read and writes by the host.



Synchronous burst writes to ADSP-21535 slaves are not supported.

Direct Writes

When a direct write to a slave DSP occurs, the address, data, and control are latched into the direct write FIFO. The FIFO supports up to eight,

64-bit wide direct writes, which can be performed with no stalls. If additional direct writes are attempted when the FIFO buffer is full, the DSP deasserts ACK (or REDY) until the buffer is no longer full. The direct write FIFO may be held off for up to four processor core cycles if all of the serial port DMA channels are active, or up to nine core cycles per chain if DMA chaining is occurring.

Direct Write Latency

The DSP handles asynchronous and synchronous direct writes differently. This difference influences the latency for the direct writes.

When a DSP bus slave receives data from an asynchronous direct write, the DSP latches the data and address in a four-level FIFO buffer. For synchronous direct writes, this buffer is two-levels deep. This buffer is called the slave write FIFO and appears in Figure 6-8 on page 6-68. In the following cycle, the slave write FIFO attempts to complete the write internally. This buffering lets the host (or DSP bus master) perform writes at the full clock rate.



The slave DSP's core cannot explicitly read the slave write FIFO. Also, the DSP cannot determine the slave write FIFO's status.

Writes to the I/O processor registers from the slave write FIFO usually occur in the following one or two cycles or when any current DMA transfer is completed. The write takes more than two cycles only if a direct write in the previous cycle was held off by a full buffer.

If the slave write FIFO is full when a write is attempted, the DSP deasserts ACK (or REDY) until the FIFO is not full. Unless higher priority on-chip DMA transfers are occurring, the slave write FIFO usually empties out within one cycle, creating a one-cycle write latency.

Slave reads are held off when there is data in the slave write FIFO—this prevents false data reads and out-of-sequence operations.

The Direct Write Pending (DWPD) bit of the SYSTAT register indicates when a direct write to internal memory is pending in the I/O processor's direct write FIFO or data is pending in the slave write FIFO (at the external port I/O pins). Direct writes and I/O processor register accesses may be completed in different sequences. If the host performs a direct memory write then writes to an I/O processor register on the DSP, the I/O processor register write may complete before the direct write.

Direct Reads

When a direct read of a DSP occurs, the address is latched on-chip by the I/O processor. If the access is asynchronous ($\overline{CS}$ asserted), REDY is deasserted asynchronously. If the access is synchronous ($\overline{CS}$ deasserted), ACK is deasserted in the following CLKIN cycle. When the data is available, the I/O processor drives the data and asserts REDY (or ACK). Direct reads cannot be pipelined like direct writes—direct reads only occur one at a time.

Direct reads have a maximum throughput of one access per every three CLKIN cycles for synchronous I/O processor register reads or have one access every four CLKIN cycles for synchronous internal memory reads. As a slave, the DSP supports synchronous burst direct read accesses, which improve throughput for internal memory reads. Maximum throughput for synchronous burst direct read accesses is summarized in Table 7-8.

Table 7-8. Direct Read Latencies—for a 1:2 Clock Ratio

Access Type	Latency (CLKIN cycles)
Single Direct Read of I/O processor register	3
Single Direct Read of internal memory	4
Burst Direct Read of I/O processor registers (EPBx only)	3-2-2-2
Burst Direct Read of Internal memory	4-2-2-2



Even with the throughput advantage of burst read transfers, direct reads are not the most efficient method of transferring data out of a slave DSP. The highest throughput is achieved by forcing the slave to become the master and having it write (or push) the data out. The advantage of direct reads is that no programming of the slave I/O processor is required.

Broadcast Writes

Broadcast writes allow simultaneous transmission of data to all of the DSPs in a multiprocessing system. The host processor (or master DSP) can perform broadcast writes to the same memory location or I/O processor register on all of the slaves. Broadcast writes can be used to implement reflective semaphores in a multiprocessing system. For more information, see "Broadcast Writes" on page 7-118.

Shadow Write FIFO

Because the DSP's internal memory must operate at high speeds, writes to the memory do not go directly into the internal memory. Instead, writes go to a two-deep FIFO called the shadow write FIFO.

When an internal memory write cycle occurs, the DSP loads the data in the FIFO from the previous write into memory, and the new data goes into the FIFO. This operation is transparent, because any reads of the last two locations written are intercepted and routed to the FIFO.



Because the ADSP-21535 DSP's shadow write FIFO automatically pushes the write to internal memory as soon as the write does not compete with a read, this FIFO's operation is completely transparent to programs. Unlike previous SHARC DSPs, there is no need for dummy writes to clear the FIFO when mixing 32- and 48-bit writes.

Data Transfers Through the EPBx Buffers

In addition to direct reads and writes, the host processor can transfer data to and from the DSP through the external port FIFO buffers, EPB0, EPB1, EPB2, and EPB3. Each of these buffers, which are part of the I/O processor register set, is an eight-location FIFO. These buffers support single-word transfers, DMA transfers, and sequential burst accesses.

DMA transfers are handled internally by the DSP's I/O processor, but single-word transfers must be handled by the processor core.

For information on external port transfers, see "External Port Channel Transfer Modes" on page 6-21. For information on external port handshaking, see "External Port Channel Handshake Modes" on page 6-22. For information on single-word interrupt-driven transfers through the external port, see "Link Port Status" on page 6-60.



To support debugging buffer transfers, the DSP has a Buffer Hang Disable (BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD discussion on page on page 6-18.

DMA Transfers

The host processor can also set up DMA transfers to and from the DSP. After the host gets control of the DSP, the host can access the on-chip

DMA control and parameter registers to set up an external port DMA operation. DMA is the most efficient way to transfer blocks of data.

- DMA Transfers to Internal Memory. The host can set up external port DMA channels to transfer data to and from DSP internal memory.
- DMA Transfers to External Memory. The host can set up an external port DMA channel to transfer data directly to external memory using the DMA request and grant lines ($\overline{\text{DMARx}}$, $\overline{\text{DMAGx}}$).

For more information, see "Setting up External Port DMA" on page 6-75.

The host may also use the $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ handshake signals for a DMA transfer as a bus slave, but may not use DMA as a bus master while $\overline{\text{HBR}}$ retains control of the bus.

Host Data Packing

The host interface uses the same data packing features as the I/O processor uses. "External Port Buffer Modes" on page 6-17. The "32-bit Data Packing" on page 7-75 and "48-Bit Instruction Packing" on page 7-79 sections describe timing for these data packing operations.

For direct reads and writes of DSP internal memory, the HPM bits determine the packing mode. For transfers to or from the EPBx data buffers, the packing mode is determined by the setting of the PMODE bits in the DMACx control register of each external port buffer.

32-bit Data Packing

For a 16-bit host bus, the DSP latches incoming data on pins DATA47-32. Similarly, the DSP drives outgoing data on DATA47-32 with the other lines

equal to zeroes. The sequence of events for 32-bit packing and unpacking is different for writes and reads.



For a 16-bit host bus, the endian format of the transfers is controlled by the HMSWF bit in the SYSCON register. If HMSWF=0, the least significant 16-bit word is packed first. If HMSWF=1, the most significant 16-bit word is packed first.

Figure 7-22 on page 7-77 shows example timing for host interface data packing.

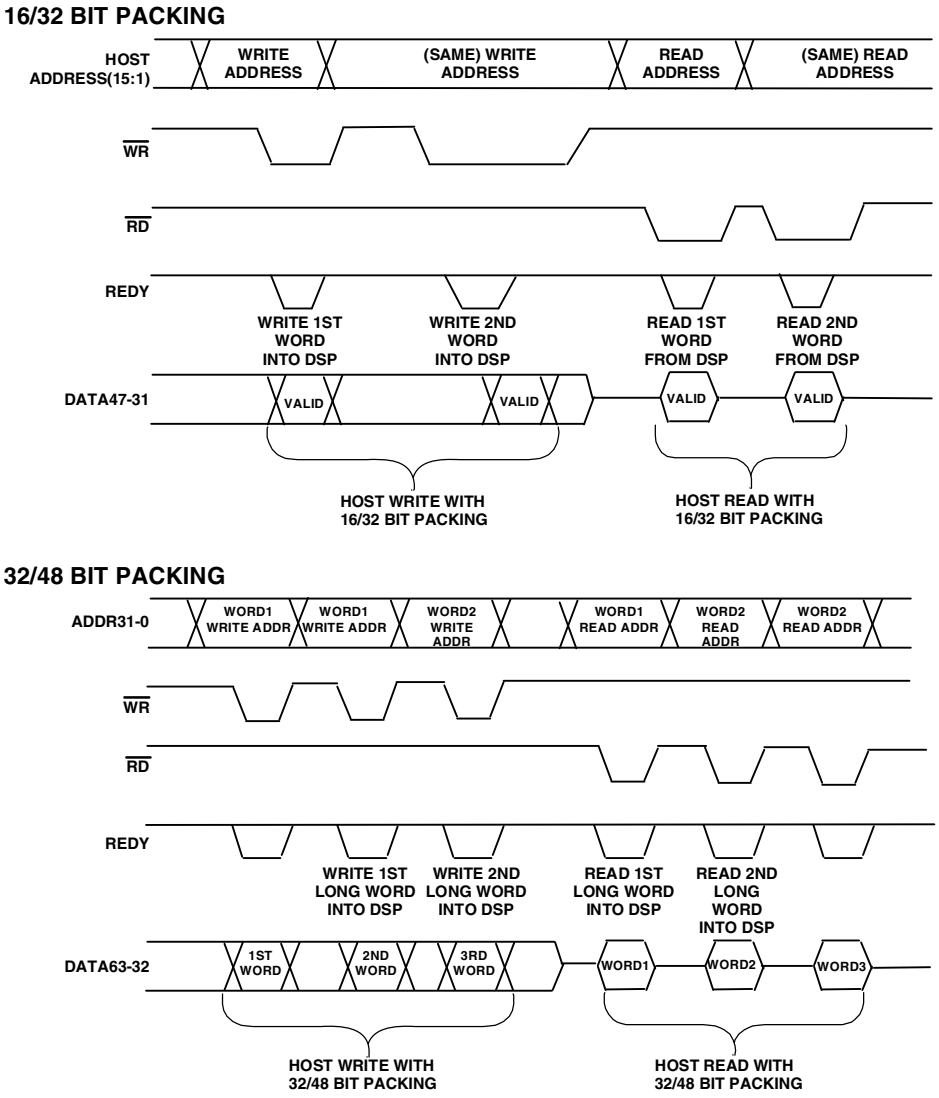


Figure 7-22. Example Timing for Host Interface Data Packing

When a host reads a 32-bit word with 16-bit unpacking using the typical bus interface hardware shown in Figure 7-24 on page 7-86, the following sequence of events occurs as illustrated in Figure 7-22 on page 7-77.

- The host initiates a read cycle by driving an address, asserting $\overline{\text{CS}}$ if the access is asynchronous, and asserting $\overline{\text{RDH}}$ (low).
- The selected DSP deasserts REDY, latches the address, and performs an internal direct read to get the data.
- When the DSP has the data, it asserts REDY and drives the first 16-bit word.
- The host latches the data and deasserts $\overline{\text{RDH}}$ (high).
- The host initiates another read access, driving the address of the data to be accessed then asserting $\overline{\text{RDH}}$.
- The DSP transmits the second 16-bit word.

When a host writes a 32-bit word with 16-bit packing using the typical bus interface hardware shown in Figure 7-24 on page 7-86, the following sequence of events occurs as illustrated in Figure 7-22 on page 7-77.

- The host initiates a write cycle by driving the write address, asserting $\overline{\text{CS}}$ if the access is asynchronous, and asserting $\overline{\text{WRH}}$ (low).
- The DSP asserts REDY when it is ready to accept data.
- The host drives the address and the first 16-bit word and deasserts $\overline{\text{WRH}}$ (high).
- The DSP latches the first 16-bit word.

- The host drives the address and initiates another write cycle for the second 16-bit word by asserting $\overline{\text{WRH}}$.
- When the DSP has accepted the second word, it performs an internal direct write to its memory (or memory-mapped I/O processor register). If the DSP's internal write has not completed by the time another host access occurs, the DSP holds off that access with REDY.

If the DSP is waiting for another 16-bit word from the host to complete the packed word, the HPS field in the SYSTAT register is non-zero. For more information, see "Host Interface Status" on page 7-81.



Because there is only one packing buffer for the host interface, the host must complete each packed transfer (or end the transfer and purge the FIFO with the HPFLSH bit) before another is begun. For more information, see "External Port Status" on page 6-57.

48-Bit Instruction Packing

The host can also download and upload 48-bit instructions over its 16- or 32-bit bus. The packing sequence for downloading DSP instructions from a 32-bit host bus (HPM=11) takes 3 cycles for every 2 words, as illustrated in Table 7-9. The 32-bit data is transferred on data bus lines 63-32 (DATA63-32). If an odd number of instruction words are transferred, the

packing buffer must be flushed by a dummy access to remove the unused word.

Table 7-9. 32-to-48-Bit Word Packing (Host Bus ↔ DSP)

	Data Bus Lines 63-48	Data Bus Lines 47-32
First transfer	Word1, bits 47-32	Word1, bits 31-16
Second transfer	Word2, bits 15-0	Word1, bits 15-0
Third transfer	Word2, bits 47-32	Word2, bits 31-16

The HMSWF bit of SYSCON is ignored for 32-to-48-bit packing.

The packing sequence for downloading or uploading DSP instructions over a 16-bit host bus takes 3 cycles for every word, as shown in Table 7-10.

Table 7-10. 16-to-48-Bit Word Packing, HMSWF=1 (Host Bus ↔ DSP)

	Data Bus Pins 47-32
First transfer	Word1, bits 47-32
Second transfer	Word1, bits 31-16
Third transfer	Word1, bits 15-0

The HMSWF bit in SYSCON determines whether the most significant 16-bit word or least significant 16-bit word is packed first.

40-bit extended precision data may be transferred using the 48-bit packing mode. For more information on memory allocation for different word widths, see “Memory Organization and Word Size” on page 5-22.

Host Interface Status

The SYSTAT register provides status information for host and multiprocessor systems. Table 7-14 on page 7-125 lists the status bits in the SYSTAT register. For more information on the SYSTAT register, see Table A-20 on page A-51.

Interprocessor Messages and Vector Interrupts

After getting control of the DSP, the host processor communicates with the DSP by writing messages to the memory-mapped I/O processor registers. In a multiprocessor system, the host can access the internal memory and I/O processor registers of every DSP.

The MSGRx registers are general-purpose registers that can be used for message passing between the host and DSP. They are also useful for semaphores and resource sharing between multiple DSPs. The MSGRx and VIRPT registers can be used for message passing in the following ways.

- Message Passing. The host can use any of the 8 message registers, MSGR0 through MSGR7, to communicate with the DSP.
- Vector Interrupts. The host can issue a vector interrupt to the DSP by writing the address of an interrupt service routine to the VIRPT register. When serviced, this high priority interrupt causes the DSP to branch to the service routine at that address.

The MSGRx and VIRPT registers also support shared-bus multiprocessing through the external port. Because these registers may be shared resources within a single DSP, conflicts may occur—your system software must prevent this. For further discussion of I/O processor register access conflicts, see "I/O Processor Registers" on page A-33.

Message Passing (MSGRx)

Three possible software protocols that the host can use for communicating with the DSP through the MSGRx message registers are: (1) vector-interrupt-driven, (2) register handshake, and (3) register write-back.

For the vector-interrupt-driven method, the host fills predetermined MSGRx registers with data and triggers a vector interrupt by writing the address of the service routine to VIRPT. The service routine should read the data from the MSGRx registers and then write "0" into VIRPT to tell the host it is done. The service routine also could use one of the DSP's FLAG3-0 pins to tell the host it has finished.

For the register handshake method, four of the MSGRx registers should be designated as follows: a receive register (R), a receive handshake register (RH), a transmit register (T), and a transmit handshake register (TH). To pass data to the DSP, the host would write data into T and then write a "1" into TH. When the DSP sees a "1" in TH, it reads the data from T and then writes back a "0" into TH. When the host sees a "0" in TH, it knows that the transfer is complete. A similar sequence of events occurs when the DSP passes data to the host through R and RH.

The register write-back method is similar to register handshaking, but uses only the T and R data registers. The host writes data to T. When the DSP sees a non-zero value in T, it retrieves it and writes back a "0" to T. A similar sequence occurs when the host is receiving data. This simpler method works well when the data to be passed does not include "0."

Host Vector Interrupts (VIRPT)

Vector interrupts are used for interprocessor commands between the host and a DSP or between two DSPs. When the external processor writes an address to the DSP's VIRPT register, the write triggers a vector interrupt. For more information, see "Multiprocessing Interrupts" on page 3-49

To use the DSP's vector interrupt feature, the host could perform the following sequence of actions.

1. Poll the DSP's VIRPT register until the host reads a certain token value (for example, zero).
2. Write the vector interrupt service routine address to VIRPT.
3. When the service routine is finished, the DSP should write the token back into VIRPT to indicate that it is finished and that another vector interrupt can be initiated.

A host using direct writes and vector interrupts should use the Direct Write Pending (DWPD) bit to determine when a direct write to internal memory is pending, because direct writes and I/O processor register accesses may be completed in different sequences. If the host performs a direct memory write to a DSP then writes to an I/O processor register on the DSP, the I/O processor register write may complete before the direct write. Because of this FIFO latency, direct writes performed just before vector interrupt writes (to VIRPT) may be delayed until after the branch to the interrupt vector.

To prevent unacceptable, latency-related sequencing, the host should check that all direct writes have completed before writing to the DSP's VIRPT register. By polling the DSP's DWPD bit and waiting for direct writes to clear, the host can write to VIRPT with correct sequencing.

System Bus Interfacing

A DSP subsystem, consisting of several DSPs with local memory may be viewed as one of several processors connected together by a system bus. Examples of such systems are the EISA bus, PCI bus, or several DSP subsystems. The processors in such a system arbitrate for the system bus using an arbitration unit. Each device on the bus that needs to become a bus master must be able to drive a bus request signal and respond to a bus

grant signal. The arbitration unit determines which request to grant in any given cycle.

Access to the DSP Bus—Slave DSP

Figure 7-23 on page 7-85 shows an example of a interface to a system bus that isolates the local DSP bus from the system bus. When the system is not accessing the DSPs, the local bus supports transfers between other local DSPs and local external memory or devices.

When the system needs to access a DSP, the system executes a read or write to the address range of the DSP subsystem. The external address comparator detects a local access and asserts $\overline{\text{HBR}}$ and one of the appropriate $\overline{\text{CS}}$ lines. The DSP holds off the system bus with REDY until the DSP is ready to accept the data. The $\overline{\text{HBG}}$ signal enables the system bus buffers. The buffers' direction for data is controlled by the read or write signals.

To avoid glitching the $\overline{\text{HBR}}$ line when addresses are changing, the address comparator may be qualified by an enable signal from the system or qualified by the system read or write signals. These methods cause $\overline{\text{HBR}}$ to be deasserted each time system read or write is deasserted or the address is changed. Because these techniques deassert $\overline{\text{HBR}}$ with each access, the overhead of an HTC occurs as part of each access. To avoid this type of overhead, systems can latch $\overline{\text{HBG}}$ during long sequences of bus accesses.

Access to the System Bus—Master DSP

Figure 7-24 on page 7-86 shows a bidirectional system interface in which the DSP subsystem can access the system bus by becoming a bus master. Before beginning the access, the DSP first requests permission to become the bus master by generating the System Bus Request signal (SBR). A bus arbitration unit determines when to respond with the System Bus Grant signal (SBG). Here, each system bus master generates and responds to its own unique pair of signals.

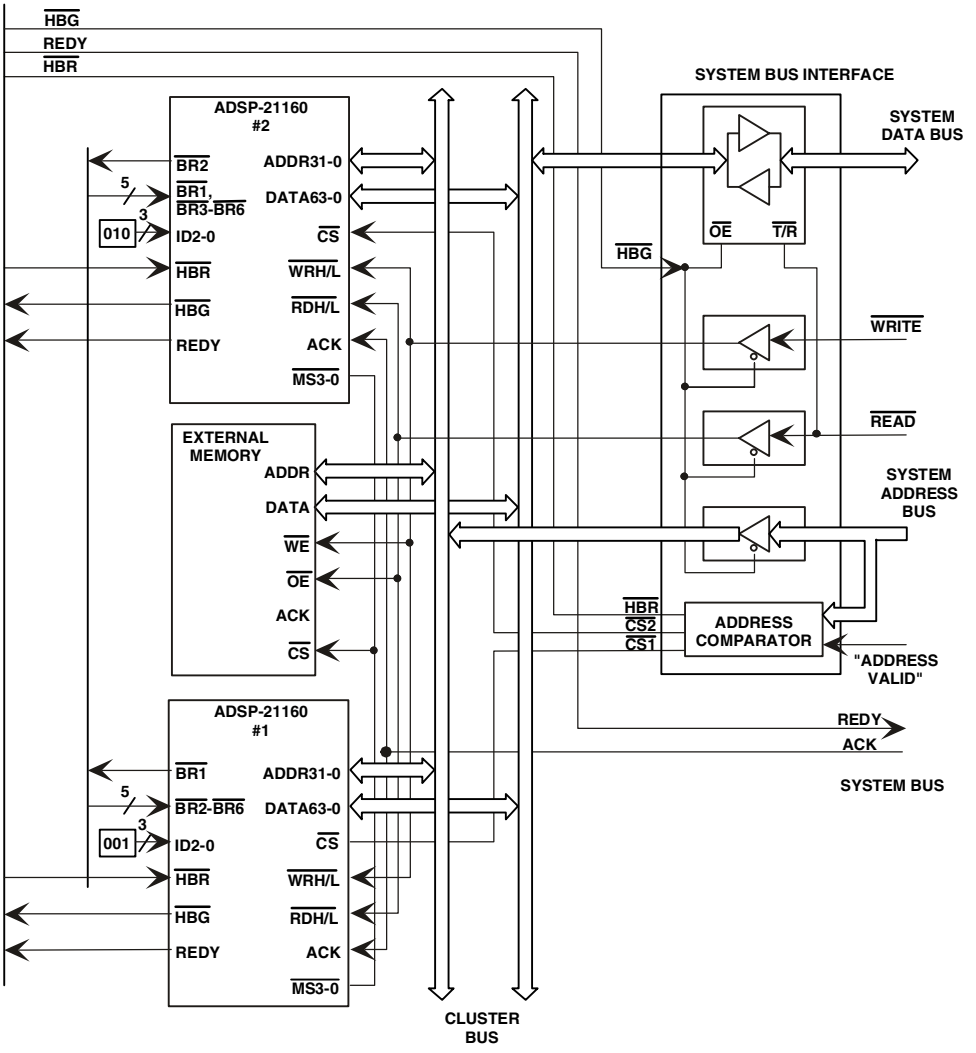


Figure 7-23. Slave DSP System Bus Interface

The method a DSP uses to arbitrate for the system bus depends on whether the access is from the DSP processor core or I/O processor. For

more information, see “Processor Core Access To System Bus” on page 7-87 and “DSP DMA Access To System Bus” on page 7-91.

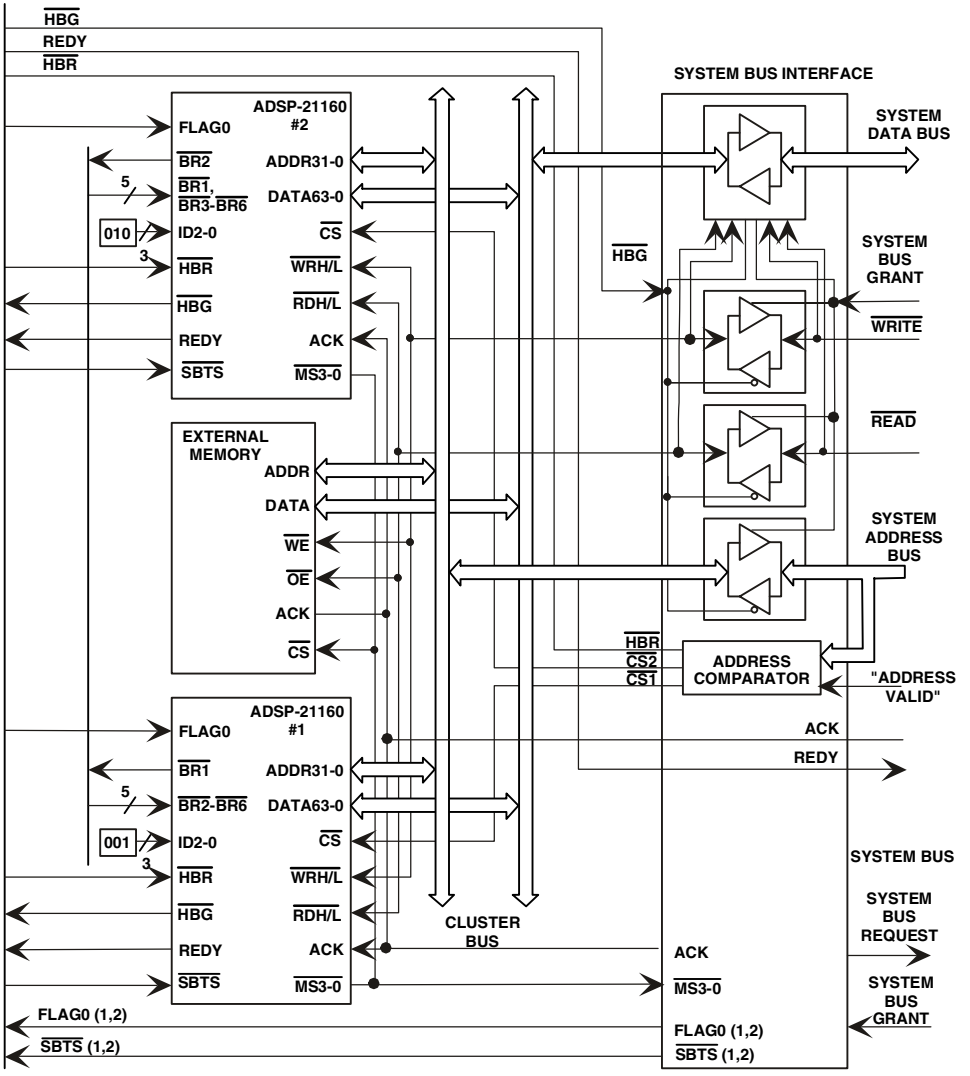


Figure 7-24. Bidirectional System Bus Interface

Processor Core Access To System Bus

The DSP core may arbitrate for the system bus by setting a flag and waiting for SBG on another flag. This technique has the benefit of not stalling the local bus while waiting. If SBG is tied to an interrupt pin, the DSP can continue processing while waiting.

Another method is for the DSP to attempt the access assuming that the system bus is available. The DSP then either waits or aborts the access if the bus is not available. The DSP begins the access to the system bus by asserting one of the memory select lines, $\overline{\text{MS3-0}}$. This assertion also asserts SBR. If the system bus is not available (for example, SBG is deasserted), the DSP should be held off with ACK. This approach is simple, but stalls the DSP and the local bus when the system bus is accessed while it is busy. To overcome this stall, programs can use the Type 10 instruction:

```
IF condition JUMP(addr), ELSE compute, DM(addr)=dreg;
```

This instruction aborts the bus access if the condition (SBG) is not true and causes a branch to a try-again-later routine. This method works well if SBG is asserted most of the time. If the Type 10 instruction is not used, a deadlock condition can result if an access is attempted before the bus is granted.



The DSP samples FLAG inputs at the CLKIN frequency, and FLAG outputs must be held stable for at least one full CLKIN cycle.

Deadlock Resolution

When both the DSP subsystem and a host processor try to access the system bus in the same cycle, a bus deadlock occurs in which neither access can complete. Normally, the master DSP responds to an $\overline{\text{HBR}}$ request by asserting $\overline{\text{HBG}}$ after the completion of the current access. If the DSP is accessing the system bus at the same time, $\overline{\text{HBG}}$ is not asserted, because this current access cannot complete. This condition results in a deadlock in which neither access can complete. The deadlock may be broken by asserting the Suspend Bus Three-state ($\overline{\text{SBTS}}$) input for one or more cycles after

the deadlock is detected—when the system bus to local bus buffer is requested from both sides.

The combination of $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ forces the master DSP to suspend its core access and assert $\overline{\text{HBG}}$. This lets the system access to the local bus proceed. The combination of $\overline{\text{HBR}}$ and $\overline{\text{SBTS}}$ should be applied only when a deadlock is caused by a DSP access to the system bus. The $\overline{\text{SBTS}}$ signal should not be used when there is a local bus transfer, because the $\overline{\text{WRH/L}}$ signal is asserted twice—once before the $\overline{\text{SBTS}}$ is asserted and once after the access resumes. For DSP-to-DSP transfers on the local bus, this double assertion violates the slave timing requirements.

The signals ACK, $\overline{\text{HBG}}$, REDY, and the data bus are all active in slave mode. If the DSP was performing an external access (which did not complete) in the same cycle that $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ were asserted, the access is suspended until $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ are both deasserted again. The timing in Figure 7-25 on page 7-89 shows example timing for the deadlock resolution case on a synchronous host interface.



As with previous SHARC DSPs, $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ can be used together for host/DSP deadlock resolution. However, if $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ are asserted while bus lock is set, the DSP three-states its bus signals, but does not go into slave mode.

When a host processor uses $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ for deadlock resolution, $\overline{\text{SBTS}}$ operates differently than when the DSP uses only $\overline{\text{SBTS}}$. For more information on how the DSP uses $\overline{\text{SBTS}}$, see the discussion on page 7-17.

The signals $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ can be used together for host/DSP core deadlock resolution but not for DMA transfers or bursts.

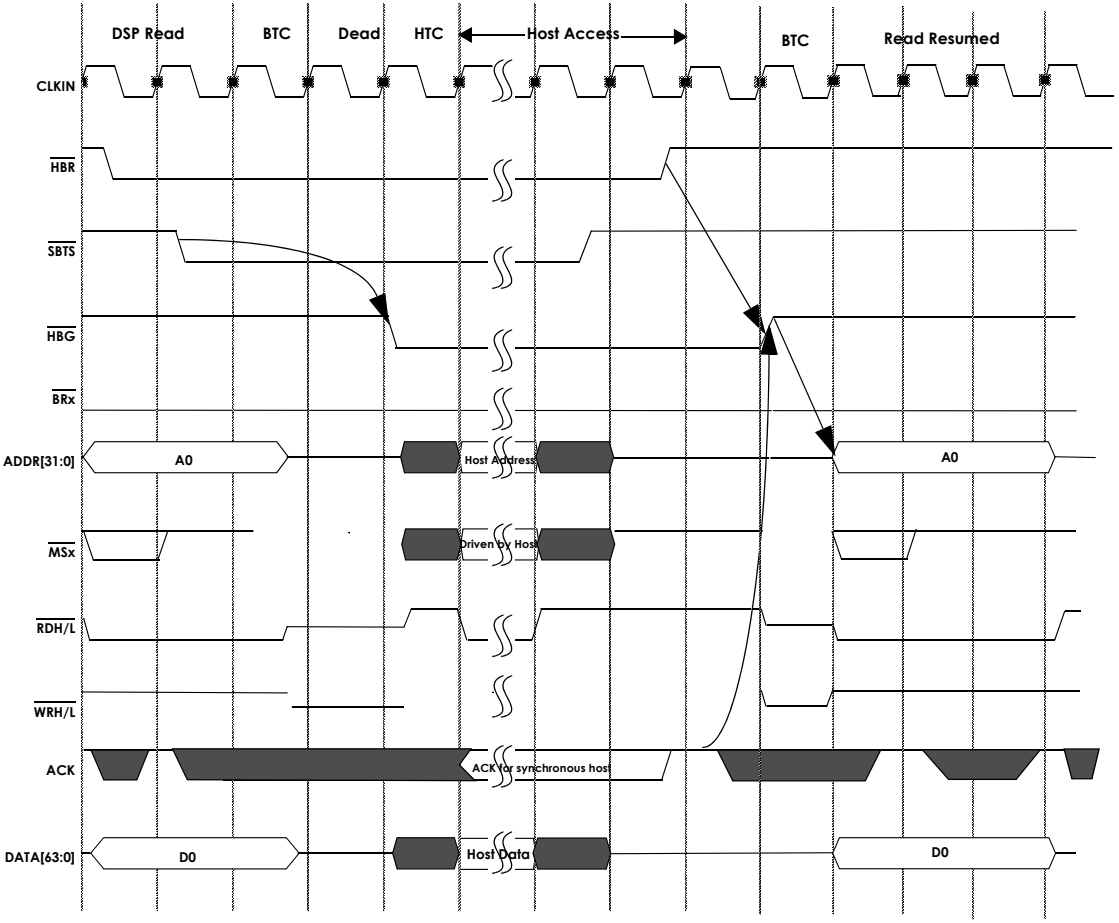


Figure 7-25. Deadlock Resolution Example Timing

The following sequence of actions allows the host processor to suspend an ongoing DSP access and gain access to its internal resources, provided that: (1) the access originates from the DSP's core, not the DMA controller, (2) a DRAM page miss is not detected for that memory access, and (3) bus lock is not enabled.

1. The host interface asserts $\overline{\text{HBR}}$.
2. The host interface determines whether the request has created a deadlock. If a deadlock is detected, the host asserts $\overline{\text{SBTS}}$ for one or more cycles.
   $\overline{\text{HBR}}$ should be asserted first. $\overline{\text{SBTS}}$ should be asserted only after detecting the deadlock. $\overline{\text{SBTS}}$ should remain asserted at least until the DSP asserts $\overline{\text{HBG}}$. $\overline{\text{SBTS}}$ should be deasserted before $\overline{\text{HBR}}$ deasserts. $\overline{\text{SBTS}}$ should be a stable signal that follows a proper setup and hold requirement as described in the *ADSP-21160 DSP Microcomputer Data Sheet*.
3. The DSP sees asserted $\overline{\text{SBTS}}$ with $\overline{\text{HBR}}$. If $\overline{\text{SBTS}}$ is asserted in cycle 0, the DSP samples it in cycle 1 and aborts the ongoing access in the same cycle.
4. The host three-states both $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ in cycle 1.
5. In cycle 3, $\overline{\text{HBG}}$ is asserted. The host acquires full control of the bus and may access any of the DSPs or peripherals on the bus. Remove $\overline{\text{SBTS}}$ before $\overline{\text{HBR}}$ is deasserted.
6. The host performs the accesses as bus master. After completing the accesses, the host deasserts $\overline{\text{HBR}}$.

7. The DSP waits for ACK to be asserted. If ACK is sampled asserted in cycle 0, $\overline{\text{HBG}}$ is deasserted in cycle 1. In cycle 2, the DSP starts the access (which it aborted to give mastership to host) as a fresh access. All the strobes fire as if it is a new access. The wait counter is also reset.

8. The DSP becomes bus master again.

   When $\overline{\text{SBTS}}$ is used for deadlock resolution, an asserted ACK is not needed to assert $\overline{\text{HBG}}$. The host, however, may need a valid ACK signal for synchronous accesses. ACK must be asserted to deassert $\overline{\text{HBG}}$ and return the bus mastership back to the DSP. ACK is then used for normal DSP transfer requirements.

DSP DMA Access To System Bus

Using the $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ inputs to resolve a system bus deadlock, as described in "Deadlock Resolution" on page 7-87, cannot be done for DMA transfers because after a DMA word transfer has begun in the DSP, it must be completed (for example, it must receive the ACK signal). If $\overline{\text{SBTS}}$ and $\overline{\text{HBR}}$ are asserted during a DMA access, the $\overline{\text{HBG}}$ pin is not asserted until the access cycle has completed. If the single DMA access is not allowed to complete, a deadlock condition may result.

To prevent system bus deadlock when using DMA, programs must make sure that SBG has been asserted before the DMA sequence begins. If a higher priority access is needed, the DMA sequence may be held off (by asserting $\overline{\text{HBR}}$) at any time after a word has been transferred. Systems must make sure that SBG is asserted before $\overline{\text{HBR}}$ is deasserted to prevent the possibility of another deadlock occurring. When the DMA sequence is complete, the DMA interrupt service routine should clear the external SBR flag.

Because the system bus is likely to be considerably slower than the DSP local bus, performance on the local bus may be considerably improved by using handshake mode DMA. In this case, the SBG signal is tied to the

DMA request line, $\overline{\text{DMARx}}$. The local and system bus accesses are only initiated when the system bus is available.



Using a FIFO in the system interface unit, to allow DMA data from the local bus to be posted, may also increase performance on the local bus when using a slow system bus.

Multiprocessing with Local Memory

Figure 7-26 shows how several DSP subsystems may be connected together on a system bus for high throughput. The gate array implements bus arbitration when the system bus is accessed. The buffers isolate the DSP local buses from the system bus.

The example system in Figure 7-26 works in the following way.

- A DSP requests the system bus with SBR when it asserts the $\overline{\text{MS2}}$ line. The gate array arbitrates between the SBR lines and then enables the highest priority group by asserting SBG, which is tied to FLAG0.
- The master DSP may connect to system memory or to other DSP groups. When the bus buffer is enabled, the read or write strobe enables should be asserted with a delay to allow the address to stabilize.
- To access a DSP slave in another group, the master DSP addresses that group's multiprocessor memory space. The gate array detects group multiprocessor memory space from three high-order address bits and asserts the $\overline{\text{HBR}}$ line for the selected group. When $\overline{\text{HBG}}$ is asserted, the gate array enables the slave's bus buffer. The high-order group address bits are cleared by the buffer to allow the

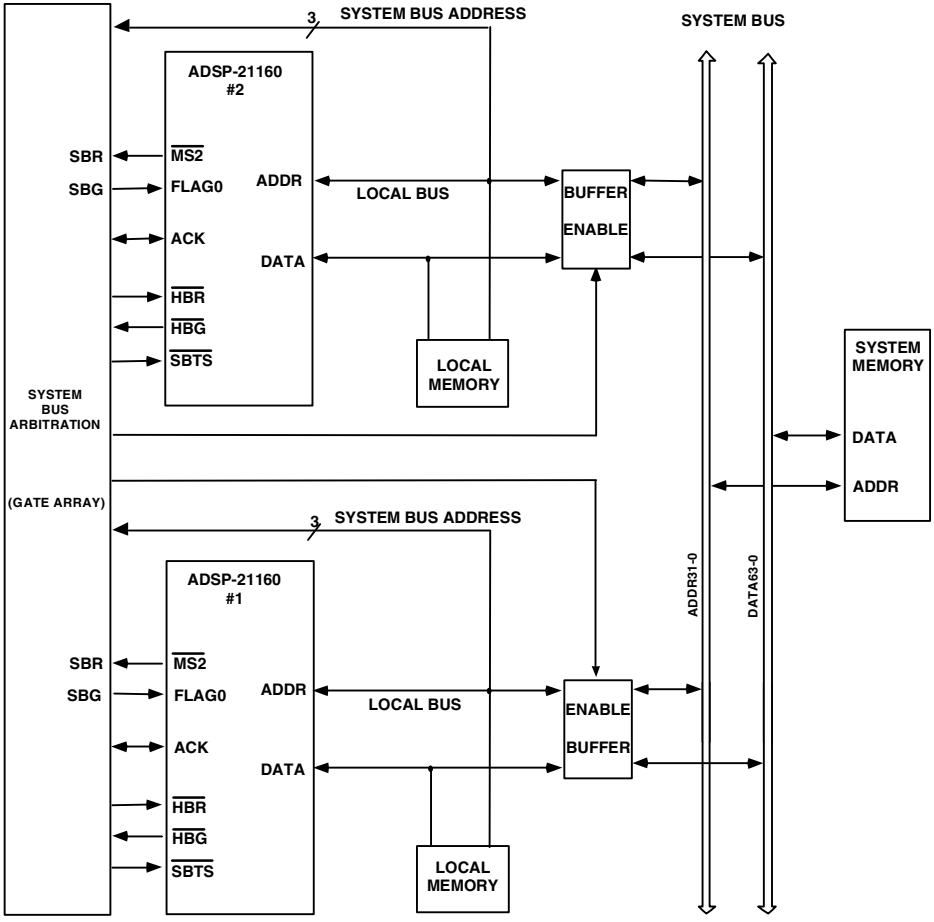


Figure 7-26. DSP Subsystems On A System Bus

group to decode the address as local multiprocessor memory space. The access is synchronous because the $\overline{CS}$ line is not asserted. The single waitstate option for the bus should be enabled.

- If two groups access each other in the same cycle, a deadlock may occur. The $\overline{SBTS}$ pin may be used to clear the deadlock.

DSP To Microprocessor Interface

A DSP without external memory may connect to a host microprocessor's bus. Depending on the microprocessor's I/O capabilities, the interface may not require any buffers. This type of connection assumes that the DSP can execute its application from internal memory most of the time and only occasionally needs to request an external access.

The host microprocessor should always keep the $\overline{\text{HBR}}$ request asserted unless it sees $\overline{\text{BR1}}$ asserted (for the $\overline{\text{BRx}}$ line of the DSP with ID=001). The host can then deassert $\overline{\text{HBR}}$ to allow the DSP to perform an external access when the host is ready to give up its bus. Usually, the host can read or write to the DSP as needed. The host accesses the DSP by asserting the $\overline{\text{CS}}$ signal and handshaking with REDY. Host Bus Grant ($\overline{\text{HBG}}$) need not be used in this system.

Multiprocessor (DSPs) Interface

The ADSP-21535 supports connecting to other ADSP-21535s to create multiprocessing DSP systems. This support includes:

- Distributed, on-chip arbitration for the shared external bus
- A unified multiprocessor address space that makes the internal memory and I/O processor registers of all DSPs directly accessible to each DSP (and host interface)
- Dedicated hardware support for interprocessor communication (for example, reflective semaphores)
- Dedicated, point-to-point communication channels between DSPs using the link ports

Figure 7-27 on page 7-96 illustrates a basic multiprocessing system. In a multiprocessor system with several DSPs sharing the external bus, any of the processors can become the bus master. The bus master has control of

the bus, which consists of the DATA63-0, ADDR31-0, and associated control lines.

Table 7-11 on page 7-95 shows the external port signals that are needed in multiprocessor DSP arbitration and communication.

Table 7-11. Signal For Cluster Multiprocessor Systems

Signal Types	Signals
Synchronization:	CLKIN, $\overline{\text{RESET}}$
Arbitration:	$\overline{\text{BR6-1}}$, $\overline{\text{PA}}$[1]
Bused Information:	ADDR31-0, DATA63-0
Master Controls:	$\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, $\overline{\text{WRL}}$, BRST
Slave Control:	ACK
Host Interface:[2]	$\overline{\text{HBR}}$, $\overline{\text{HBG}}$, $\overline{\text{CS}}$[3], REDY[3], $\overline{\text{SBTS}}$

1 Optional, only needed if Priority Access function is used
2 Optional, only needed if Host Interface is used.
3 Optional, only needed if asynchronous Host Interface employed.

The internal memory and I/O processor registers of the system's DSPs comprise the multiprocessor memory space. Multiprocessor memory space is mapped into the unified address space of each DSP. For more information, see the multiprocessor memory map in Figure 5-8 on page 5-18.

After a DSP becomes the bus master, it can directly read and write the internal memory of any other slave DSP. The master can also read and write to any of the slave's I/O processor registers, including their external port FIFO data buffers. For example, the master DSP may write to a slave's I/O processor registers to set up DMA transfers or to send a vector interrupt.

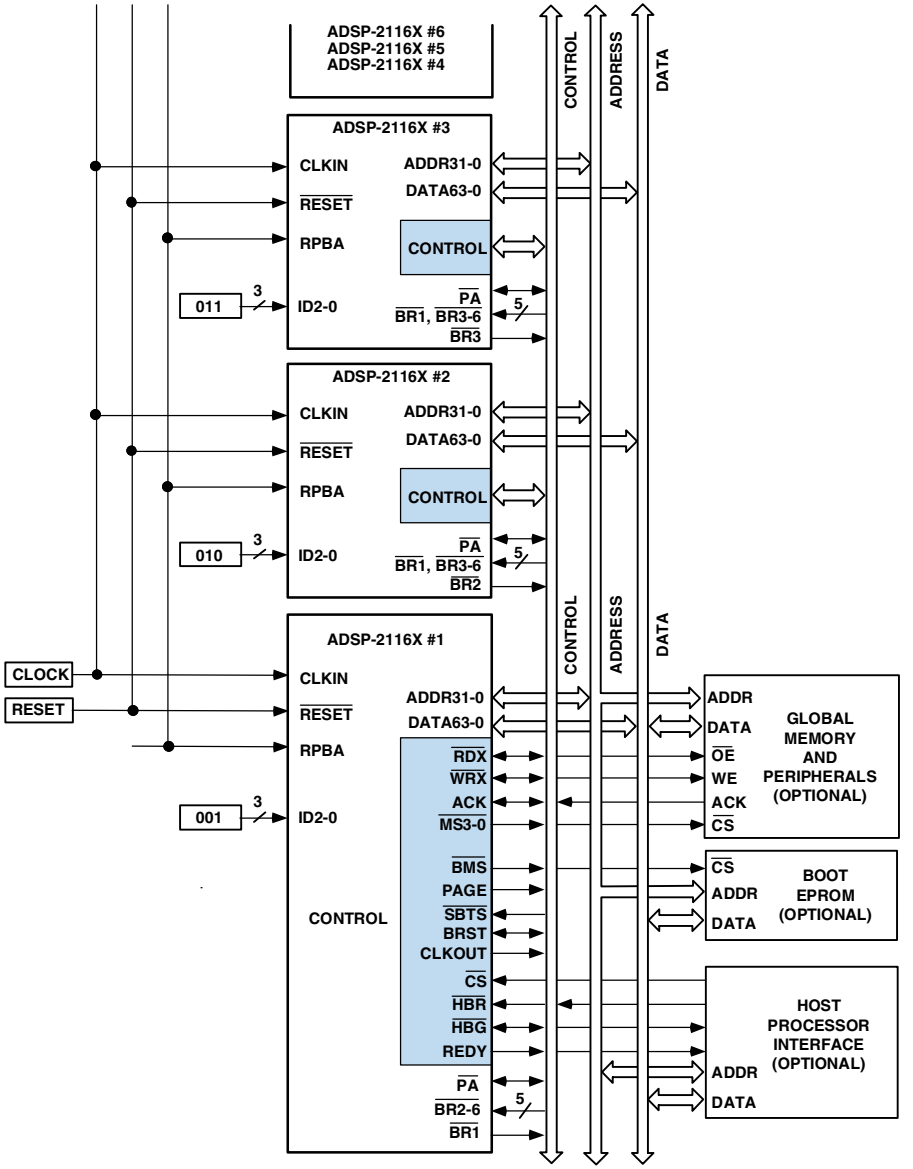


Figure 7-27. ADSP-21535 Multiprocessor System

Multiprocessing System Architectures

Multiprocessor systems typically use one of two schemes to communicate between processor nodes. One scheme uses dedicated point-to-point communication channels. In the other scheme, nodes communicate through a single shared global memory over a parallel bus.

The DSP supports point-to-point communication—data flow multiprocessing—through its six link ports. Also, the DSP supports a shared parallel bus communication—cluster multiprocessing—through its link ports and external port. For more information on data flow multiprocessing, see "Data Flow Multiprocessing" below and "Data Flow Multiprocessing With Link Ports" on page 8-22. For more information on cluster multiprocessing, see "Cluster Multiprocessing" on page 7-98.

Data Flow Multiprocessing

Data flow multiprocessing works for applications requiring high computational bandwidth, but requiring only limited flexibility. The program partitions its algorithm sequentially across multiple processors and passes data through a line of processors, as shown in Figure 7-27.

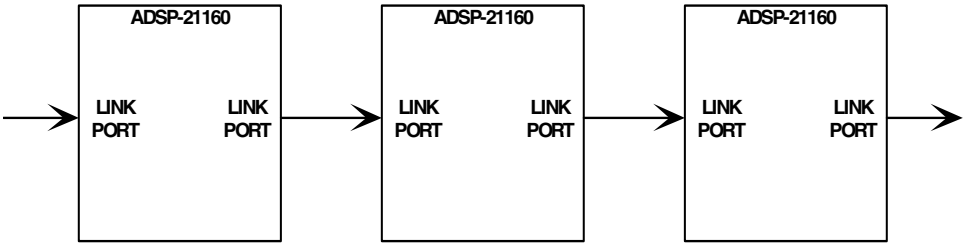


Figure 7-28. Data Flow Multiprocessing

The DSP provides complete support for data flow multiprocessing applications, because the DSP eliminates the need for interprocessor data FIFOs and external memory. The internal memory of the DSP is usually large enough to contain both code and data for most applications using data-flow system-topology. All that a data flow system requires are a num-

ber of DSPs and point-to-point signals connecting them. This design yields savings in complexity, board space, and system cost. For more information on connecting multiple DSPs using link ports, see "Host Processor Access To Link Buffers" on page 8-12.

Cluster Multiprocessing

Cluster multiprocessing works for applications where a fair amount of flexibility is required. This flexibility is needed when a system must be able to support a variety of different tasks, some of which may be running concurrently. The cluster multiprocessing configuration is shown in Figure 7-29. Also, the DSP has an on-chip host interface that lets a cluster be interfaced to a host processor or another cluster.

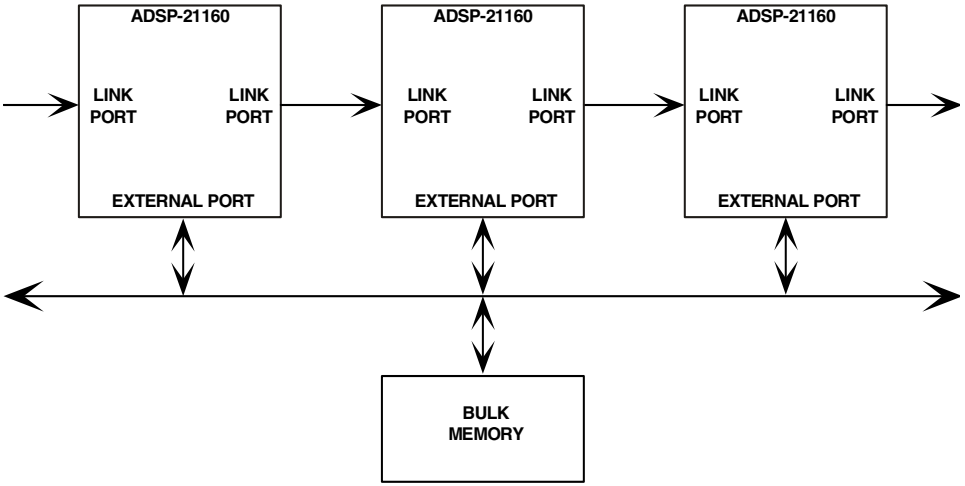


Figure 7-29. Cluster Multiprocessing

Cluster multiprocessing systems include multiple DSPs connected to a parallel bus that supports interprocessor access of on-chip memory and access to shared global memory. In a typical cluster of DSPs, up to six processors and a host can arbitrate for the bus. The on-chip bus arbitration logic lets these processors share the common bus.

The DSP's features (such as large internal memory, link ports, and external port FIFOs) help eliminate the need for any extra hardware in the cluster multiprocessor configuration. External memory—local and global—can frequently be eliminated in this type of system.

The DSP supports fixed and rotating priority schemes. Other supported techniques include bus locking, timed release, DMA prioritization, and core processor access preemption of background DMA transfers. The on-chip arbitration logic lets transitions in bus mastership take up to only one cycle of overhead. Bus requests are generated implicitly when a processor accesses an external address. Because each processor monitors all bus requests and applies the same priority logic to the requests, each can independently determine who is the next bus master.

After getting mastership of the bus, a DSP can access external memory and the internal memory and I/O processor registers of all other DSPs (slaves) in the system. A DSP can directly transfer data to another DSP or set up a DMA channel to transfer the data. The DSPs are mapped into a common memory map—to identify the address space of each DSP within the unified memory map of the system cluster. Also, each DSP has a unique `ID`. The DSP's I/O processor registers, internal memory, and external memory are all part of the unified address space. This shared on-chip memory eliminates the need to use external memory for message passing between DSPs and simplifies software communications. DSPs can write directly into each other's memory, saving an extra transfer step.

The multiprocessor communication between DSPs is eased with the broadcast write feature, which lets a DSP write to all processors simultaneously. This broadcast supports reflective semaphores, where a processor polls its own internal copy of the semaphore and only uses the external bus for a broadcast write to all other processors when it wants to change it. This reduces communications traffic on the external bus.

The cluster configuration allows the DSPs to have a very fast node-to-node data transfer rate. Clusters also allow a simple, efficient software-communication model.

For example, all of the required setup operations for a DMA transfer can be accomplished by a single DSP on one side of the transfer. The other processor is not interrupted until the DMA transfer is complete.



The DSP's internal memory facilitates I/O in multiprocessor systems. The on-chip, dual-ported RAM supports full-speed inter-DSP transfers concurrent with dual accesses by the DSP's processor core. No cycles are stolen from the processor core, and the processor's full performance is maintained during these accesses.

Link Port Data Transfers In A Cluster. A bottleneck exists within the cluster because only two DSPs can communicate over the shared bus during each cycle—other DSPs are held off until the bus is released. Because the DSP can also perform point-to-point link port transfers within a cluster, systems can eliminate this bottleneck by setting up data communication through the link ports. Data links between DSPs can be dynamically set up and initiated over the common bus. All six link ports can operate simultaneously on each DSP.

A disadvantage of the link ports is that individual transfers occur at a much lower rate than that of the shared parallel bus. Because the link ports' 8-bit data path is smaller than the processor's native word size, the transfer of each word requires multiple clock cycles. Link ports may also require more software overhead and complexity because they must be set up on both sides of the transfers before they can occur.

SIMD Multiprocessing. For certain classes of applications such as radar imaging, a Single-Instruction, Multiple-Data (SIMD) array of DSPs may be the most efficient topology to coordinate a large number of DSPs in a single system. The SIMD array of Figure 7-29 on page 7-98 consists of multiple DSPs connected in a two- or three-dimensional mesh. The data link ports provide nearest neighbor communications and through-routing of data. A single master DSP provides the instruction stream that the array executes. Data flow in and out the array can be managed through multiple serial port streams.

Multiprocessor Bus Arbitration

Multiple DSPs can share the external bus with no additional arbitration logic. Arbitration logic is included on-chip to allow the connection of up to six DSPs and a host processor. The DSP accomplishes bus arbitration through the $\overline{\text{BR1-6}}$, $\overline{\text{HBR}}$, and $\overline{\text{HBG}}$ signals. $\overline{\text{BR1-6}}$ arbitrate between multiple DSPs, and $\overline{\text{HBR}}$/$\overline{\text{HBG}}$ pass control of the bus from the DSP bus master to the host and back. The priority scheme for bus arbitration is determined by the setting of the RPBA pin.

Table 7-12 defines the DSP pins used in multiprocessing systems.

Table 7-12. Multiprocessing DSP Pins

Signal	Type	Definition
$\overline{\text{BR6-1}}$	I/O/S	Multiprocessing Bus Requests. Used by multiprocessing DSPs to arbitrate for bus mastership. A DSP only drives its own $\overline{\text{BRx}}$ line (corresponding to the value of its ID2-0 inputs) and monitors all others. In a multiprocessor system with less than six DSPs, the unused $\overline{\text{BRx}}$ pins should be tied high; the processor's own $\overline{\text{BRx}}$ line must not be tied high or low because it is an output. Note that ID=00x device enables keeper latch/pull-up devices on certain signals. For more information, see Table 11-1 on page 11-3.
ID2-0	I	Multiprocessing ID. Determines which multiprocessing bus request ($\overline{\text{BR1}}$—$\overline{\text{BR6}}$) is used by DSP. ID=001 corresponds to $\overline{\text{BR1}}$, ID=010 corresponds to $\overline{\text{BR2}}$, etc. (ID=000 is used in single-processor systems.) These lines are a system configuration selection which should be hardwired or only changed at reset.
I = Input, S = Synchronous, (o/d) = Open Drain; O = Output, A = Asynchronous, (a/d) = Active Space		

Table 7-12. Multiprocessing DSP Pins (Cont'd)

Signal	Type	Definition
RPBA	I	Rotating Priority Bus Arbitration Select. When RPBA is high, rotating priority for multiprocessor bus arbitration is selected. When RPBA is low, fixed priority is selected. This signal is a system configuration selection which must be set to the same value on every DSP. If the value of RPBA is changed during system operation, it must be changed in the same CLKIN cycle on every DSP.
$\overline{\text{PA}}$	(a/d) I/O/S	Priority Access. The DSP slave may assert the $\overline{\text{PA}}$ signal to interrupt background DMA transfers and gain access to the external bus. This signal is asserted when a DSP slave's processor core requests the bus or if an external DMA channel requests the bus with the DMACx PRIO control bit set. The $\overline{\text{PA}}$ signal is an active drive output, which may be asserted (low) by one or more slaves. It is deasserted (high) by the master. A protocol is employed to avoid driver contention.
I = Input, S = Synchronous, (o/d) = Open Drain; O = Output, A = Asynchronous, (a/d) = Active Space		

The ID2-0 pins provide a unique identity for each DSP in a multiprocessing system. The first DSP should be assigned ID=001, the second should be assigned ID=010, and so on. One of the DSPs must be assigned ID=001 in order for the bus synchronization scheme to function properly.



The DSP with ID=001 holds the external bus control lines stable during reset.

When the ID2-0 inputs of a DSP are equal to 001, 010, 011, 100, 101, or 110, the DSP configures itself for a multiprocessor system and maps its internal memory and I/O processor registers into the multiprocessor memory space. ID=000 configures the DSP for a single-processor system. ID=111 is reserved and should not be used.

A DSP in a multiprocessor system can determine which processor is the current bus master, by reading the CRBM2-0 bits of the SYSTAT register. These bits give the value of the ID2-0 inputs of the current bus master.

Conditional instructions can be written, depending on whether the DSP is the current bus master in a multiprocessor system. The assembly language mnemonic for this condition code is Bm, and its complement is Not Bm (not bus master). The Bm condition indicates whether the DSP is the current bus master. For more information, see "Conditional Sequencing" on page 3-54. To use the bus master condition, the condition code select (CSEL) field in the MODE1 register must be zero or the condition is always evaluated as false.

Bus Arbitration Protocol

The Bus Request ($\overline{\text{BR1-6}}$) pins are connected between each DSP in a multiprocessing system, with the number of $\overline{\text{BRx}}$ lines used equal to the number of DSPs in the system. Each processor drives the $\overline{\text{BRx}}$ pin that corresponds to its ID2-0 inputs and monitors all others. If less than six DSPs are used in the system, the unused $\overline{\text{BRx}}$ pins should be tied high.

When one of the slave DSPs needs to become bus master, it automatically initiates the bus arbitration process by asserting its $\overline{\text{BRx}}$ line at the beginning of the cycle. Later in the same cycle, the DSP samples the value of the other $\overline{\text{BRx}}$ lines.

The cycle in which mastership of the bus is passed from one DSP to another is called a Bus Transition Cycle (BTC). A bus transition cycle occurs when the current bus master's $\overline{\text{BRx}}$ pin is deasserted and one or more of the slave's $\overline{\text{BRx}}$ pins is asserted. The bus master can retain bus mastership by keeping its $\overline{\text{BRx}}$ pin asserted. Also, the bus master does not always lose bus mastership when it deasserts its $\overline{\text{BRx}}$ line—another $\overline{\text{BRx}}$ line must be asserted by one or more of the slaves at the same time. In this case, when no other $\overline{\text{BRx}}$ is asserted, the master does not lose any bus cycles.

By observing all of the $\overline{\text{BRx}}$ lines, each DSP can detect when a bus transition cycle occurs and which processor has become the new bus master. A bus transition cycle is the only time that bus mastership is transferred.

After conditions determine that a bus transition cycle is going to occur, every DSP in the system evaluates the priority of the $\overline{\text{BRx}}$ lines asserted within that cycle. For a description of bus arbitration priority, see "Bus Arbitration Priority (RPBA)" on page 7-107. The DSP with the highest priority request becomes the bus master on the following cycle, and all of the DSPs update their internal record to indicate which DSP is the current bus master. This information can be read from the current bus master field, CRBM, of the SYSTAT register. Figure 7-30 shows typical timing for bus arbitration.

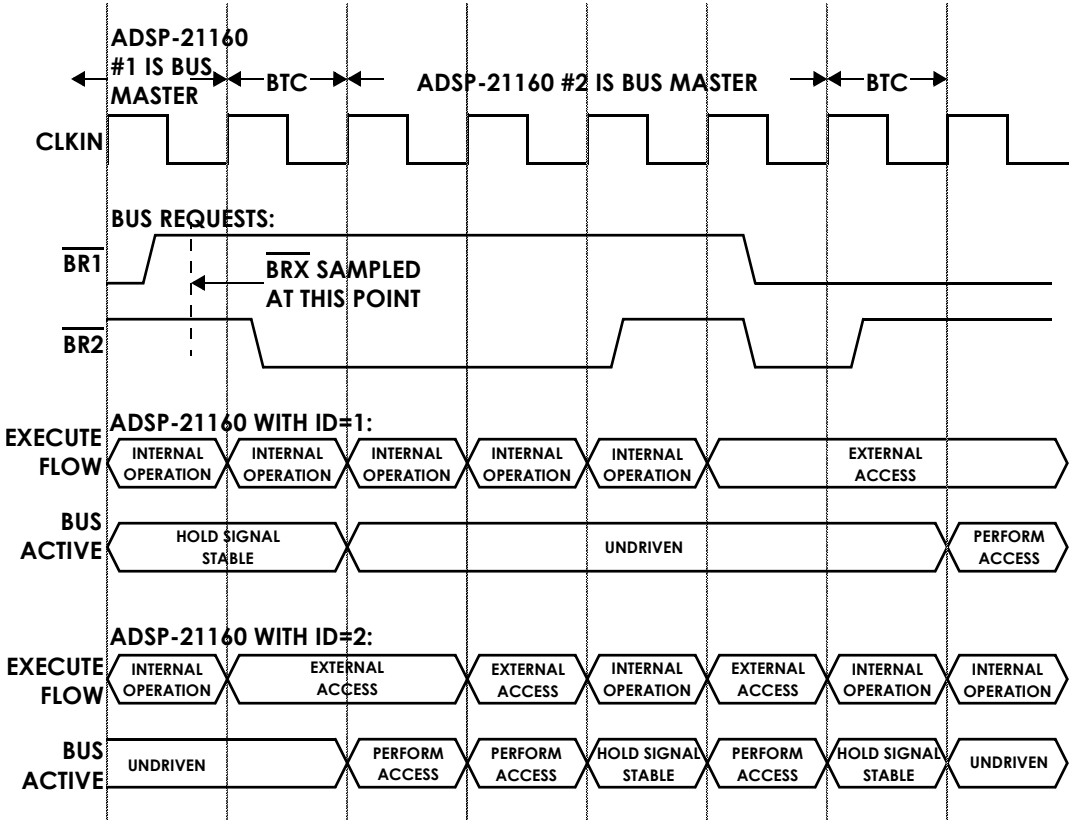


Figure 7-30. Bus Arbitration Timing

The actual transfer of bus mastership is accomplished by the current bus master three-stating the external bus—DATA63-0, ADDR31-0, $\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, $\overline{\text{WRL}}$, BRST, $\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, PAGE, $\overline{\text{HBG}}$, $\overline{\text{DMAG2-1}}$—at the end of the bus transition cycle and the new bus master driving these signals at the beginning of the next cycle. The bus strobes ($\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, and $\overline{\text{WRL}}$) and $\overline{\text{MS3-0}}$ are driven high (inactive) before three-stating occurs. ACK must be sampled high by the new master before it starts a new bus operation. For more information, see Figure 7-31 on page 7-106.

During bus transition cycle delays, execution of external accesses are delayed. When one of the slave DSPs needs to perform an external read or write, it automatically initiates the bus arbitration process by asserting its $\overline{\text{BRx}}$ line. This read or write is delayed until the processor receives bus mastership. If the read or write was generated by the DSP's processor core (not the I/O processor), program execution stops on that DSP until the instruction is completed.

The following steps occur as a slave acquires bus mastership and performs an external read or write over the bus as shown in Figure 7-31.

1. The slave determines that it is executing an instruction which requires an off-chip access. It asserts its $\overline{\text{BRx}}$ line at the beginning of the cycle. Extra cycles are generated by the core processor (or I/O processor) until the slave acquires bus mastership.
2. To acquire bus mastership, the slave waits for a bus transition cycle in which the current bus master deasserts its $\overline{\text{BRx}}$ line. If the slave has the highest priority request in the bus transition cycle, it becomes the bus master in the next cycle. If not, it continues waiting.
3. At the end of the bus transition cycle the current bus master releases the bus, and the new bus master starts driving.

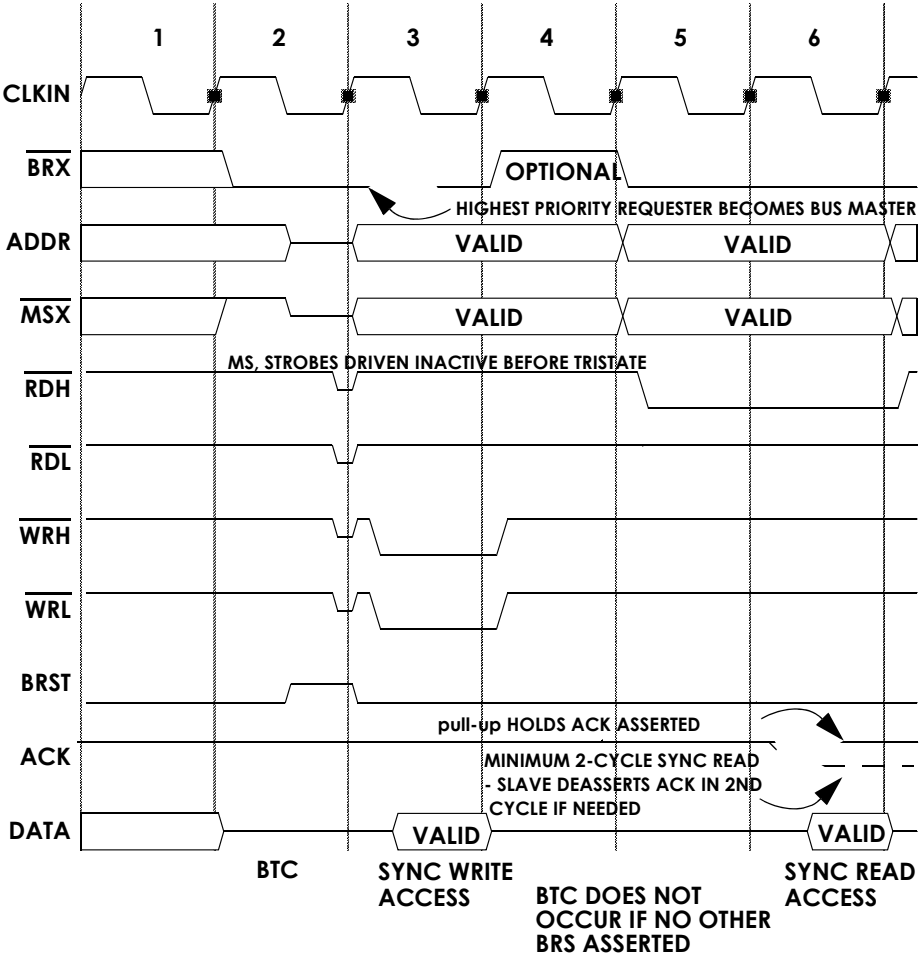


Figure 7-31. Bus Request and Read/Write Timing

During the CLKIN cycle in which the bus master deasserts its $\overline{\text{BRx}}$ output, it three-states its outputs in case another bus master wins arbitration and enables its drivers in the next CLKIN cycle. If the current bus master retains

control of the bus in the next cycle, it enables its bus drivers, even if it has no bus operation to run.

The DSP with ID=00x enables internal keeper latches, or pull-up devices, on key signals, including the address and data buses, strobes, and ACK. These devices provide a weak current source or sink—approximate 20KΩ impedance—to keep these signals from drifting near input receiver thresholds when all drivers are three-stated. For more information, see Table 11-1 on page 11-3.

When the bus master stops using the bus, its $\overline{\text{BRx}}$ line is deasserted, allowing other DSPs to arbitrate for mastership if they need it. If no other DSPs are asserting their $\overline{\text{BRx}}$ line when the master deasserts its $\overline{\text{BRx}}$, the master retains control of the bus and continues to drive the memory control signals until: 1) it needs to use the bus again, or 2) another DSP asserts its $\overline{\text{BRx}}$ line.

While a slave waits to be a master for a DMA transfer, it asserts $\overline{\text{BRx}}$. If that slave's core accesses the DA group registers, the $\overline{\text{BRx}}$ is deasserted during that access.

Bus Arbitration Priority (RPBA)

To resolve competing bus requests, there are two available priority schemes: fixed and rotating. The RPBA pin selects the scheme. When RPBA is high, rotating priority bus arbitration is selected, and when RPBA is low, fixed priority is selected.

The RPBA pin must be set to the same value on each DSP in a multiprocessing system. If the value of RPBA is changed during system operation, it must be changed synchronously to CLKIN and must meet a setup time that lets all DSPs recognize the change in the same cycle. The priority scheme changes in that (same) cycle.

In the fixed priority scheme, the DSP with the lowest ID number among the competing bus requests becomes the bus master. If, for example, the

processor with ID=010 and the processor with ID=100 request the bus simultaneously, the processor with ID=010 becomes bus master in the following cycle.



Each DSP knows the ID of the other processors requesting the bus, because their ID corresponds to the $\overline{\text{BRx}}$ line being used.

The rotating priority scheme gives roughly equal priority to each DSP. When rotating priority is selected, the priority of each processor is reassigned after every transfer of bus mastership. Highest priority is rotated from processor to processor as if they were arranged in a circle—the DSP located next to (one place down from) the current bus master is the one that receives highest priority. Table 7-13 shows an example of how rotating priority changes on a cycle-by-cycle basis.

Table 7-13. Rotating Priority Arbitration Example

Cycle Number	Hardwired Processor IDs and Priority[1]					
	ID1	ID2	ID3	ID4	ID5	ID6
1[2]	M	1	2-$\overline{\text{BR}}$	3	4	5
2	4	5-$\overline{\text{BR}}$	M-$\overline{\text{BR}}$	1	2	3
3	4	5-$\overline{\text{BR}}$	M	1	2	3
4	5-$\overline{\text{BR}}$	M	1	2	3	4-$\overline{\text{BR}}$
5[3]	1-$\overline{\text{BR}}$	2	3	4	5	M

1 The following symbols appear in these cells: 1-5 = assigned priority, M = bus mastership (in that cycle), $\overline{\text{BR}}$ = requesting bus mastership with $\overline{\text{BRx}}$
2 Initial priority assignments
3 Final priority assignments

Mastership Timeout Bus

In either the fixed or rotating priority scheme, systems may need to limit how long a bus master can retain the bus. Systems can limit bus mastership by forcing the bus master to deassert its $\overline{\text{BRx}}$ line after a specified

number of CLKIN cycles and giving the other processors a chance to acquire bus mastership.

To setup a bus master timeout, a program must load the BMAX register with the maximum number of CLKIN cycles (minus 2) that the DSP can retain bus mastership:

BMAX = (maximum # of bus mastership CLKIN cycles) – 2



Internal processor clock cycles are a multiple of CLKIN cycles.

The minimum value for BMAX is 2, which lets the processor retain bus mastership for 4 CLKIN cycles. Setting BMAX=1 is not allowed. To disable the bus master timeout function, set BMAX=0.

Each time a DSP acquires bus mastership, its BCNT register is loaded with the value in BMAX. BCNT is then decremented in every CLKIN cycle that the master performs a read or write over the bus and any other (slave) DSPs are requesting the bus. Any time the bus master deasserts its $\overline{\text{BRx}}$ line, BCNT is reloaded from BMAX.

When BCNT decrements to zero, the bus master first completes its off-chip read/write and then deasserts its own $\overline{\text{BRx}}$ (any new off-chip accesses are delayed)—this allows transfer of bus mastership. If the ACK signal is holding off an access when BCNT reaches zero, bus mastership is not relinquished until the access can complete.

If BCNT reaches zero while a burst transfer is in progress, the bus master completes the burst transfer before deasserting its $\overline{\text{BRx}}$ output. If BCNT reaches zero while bus lock is active, the bus master does not deassert its $\overline{\text{BRx}}$ line until bus lock is removed. If $\overline{\text{HBR}}$ is being serviced, BCNT stops decrementing and continues only after $\overline{\text{HBR}}$ is deasserted.



Bus lock is enabled by the BUSLK bit in the MODE2 register. For more information, see "Bus Lock and Semaphores" on page 7-121.

Priority Access

The Priority Access signal ($\overline{\text{PA}}$) lets external bus accesses by a slave DSP take priority over ongoing DMA transfers. Normally when external port DMA transfers are in progress, the slave DSPs cannot use the external bus until the DMA transfer is finished. By asserting its $\overline{\text{PA}}$ pin, the slave DSP can acquire the bus without waiting for the DMA operation to complete. The $\overline{\text{PA}}$ signal can also be asserted by a slave with a high-priority DMA access pending on the external bus.

If the $\overline{\text{PA}}$ signal is not used in a multiprocessor system, the DSP bus master does not give up the bus to another DSP until: (1) a cycle in which it does not perform an external bus access or (2) a bus timeout. If a slave DSP needs to send a high priority message or perform an important data transfer, it normally must wait until any DMA operation completes. Using the $\overline{\text{PA}}$ signal lets the slave perform its higher priority bus access with less delay.

Each of the DMACx registers has a PRIO bit that raises that DMA channel to a higher priority than all other internal DMA channels that do not have the PRIO bit set. Unless configured differently with the EBPR bit in the SYSCON register, this channel still has lower priority (internally) than the core. Programs should be careful to minimize the number of DMA channels enabled to high priority status in the multiprocessor system, because both core and (external) high priority DMA requests from slaves are arbitrated at the same priority level.

For example, a slave core cannot arbitrate bus ownership away from a high priority DMA transfer unless the bus timeout (BMAX function) occurs.

When $\overline{\text{PA}}$ is asserted, the current DSP bus master deasserts its $\overline{\text{BRx}}$ output, and gives up the bus, provided:

- Its core does not have an external access pending, and
- None of its external bus DMA channels have pending high-priority bus requests.

All DSP slaves also deassert their $\overline{\text{BRx}}$ outputs, if each slave meets the same provisions. The current bus master never asserts $\overline{\text{PA}}$, because it already has control of the bus. If the current master detects a condition that would assert $\overline{\text{PA}}$ while it is bus master, it performs that high priority operation before giving up bus ownership.

In the CLKIN cycle after $\overline{\text{PA}}$ has been asserted, only the DSP slaves with a pending high priority access have their bus requests asserted. Bus arbitration proceeds as usual with the highest priority device becoming the master when the previous bus master releases its $\overline{\text{BRx}}$ output.

The new master samples all $\overline{\text{BRx}}$ inputs after gaining bus mastership—during the cycle that follows the BTC. If no other bus requests are asserted, the master is the only device driving $\overline{\text{PA}}$, and the master deasserts and three-state $\overline{\text{PA}}$ in this cycle as shown in Figure 7-32.

If the master samples other $\overline{\text{BRx}}$ inputs as asserted, multiple devices are driving $\overline{\text{PA}}$, and the new bus master cannot deassert $\overline{\text{PA}}$. The new bus master three-states its $\overline{\text{PA}}$ driver in this case. All DSP slaves recognize the cycle following the BTC. They do not assert $\overline{\text{PA}}$ during this cycle, unless they were already driving their $\overline{\text{BR}}$ and $\overline{\text{PA}}$ outputs in the BTC.

This behavior is demonstrated in Figure 7-33.

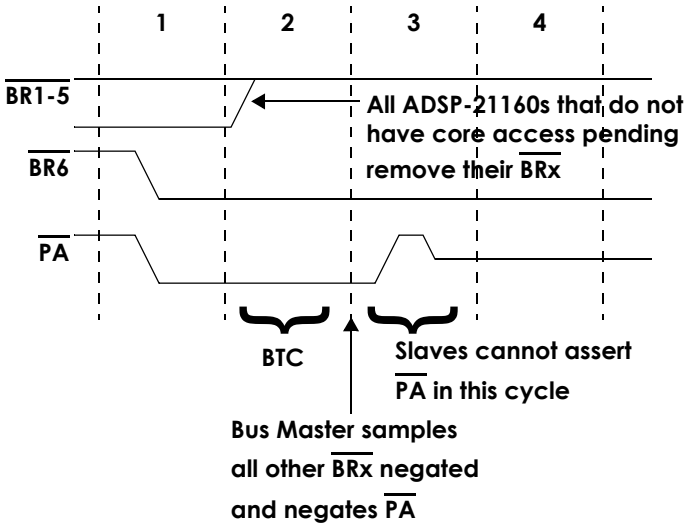


Figure 7-32. Example PA Deassertion

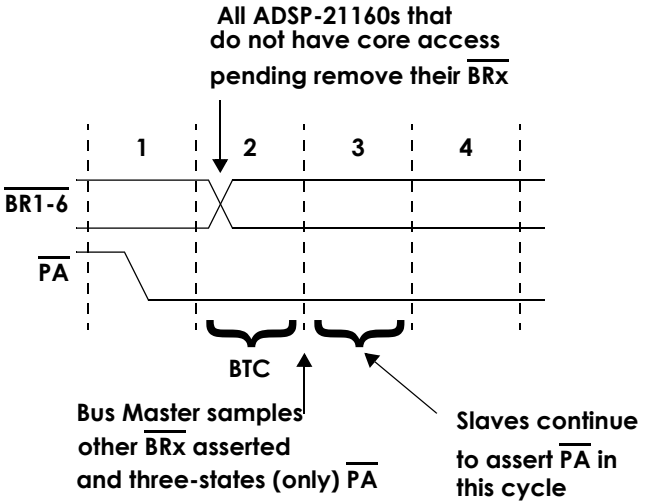


Figure 7-33. Example of PA Driven by Multiple Slaves

Bus Synchronization After Reset

When a multiprocessing system is reset ($\overline{\text{RESET}}$ asserted), the bus arbitration logic on each processor must synchronize, making sure that only one DSP drives the external bus. One DSP must become the bus master, and all other processors must recognize which one it is before actively arbitrating for the bus. The bus synchronization scheme also lets the system safely bring individual DSPs into and out of reset.



The only difference between the soft and hard reset is that the external bus arbitration does not get affected by a soft reset (no bus synchronization at soft reset). The PLL also does not get reset at soft reset.

One of the DSPs in the system must be assigned ID=001 in order for the bus synchronization scheme to function properly. This processor also holds the external bus control lines stable during reset. Bus arbitration synchronization is disabled if the DSP is in a single-processor system (ID=000).

To synchronize their bus arbitration logic and define the bus master after a system reset, the multiple DSPs obey the following rules:

- All DSPs except the one with ID=001 deassert their $\overline{\text{BRx}}$ line during reset. They keep their $\overline{\text{BRx}}$ deasserted for at least two cycles after reset and until their bus arbitration logic is synchronized.
- After reset, a DSP considers itself synchronized when it detects a cycle in which only one $\overline{\text{BRx}}$ line is asserted. The DSP identifies the bus master by recognizing which $\overline{\text{BRx}}$ is asserted and updates its internal record to indicate the current master.
- The DSP with ID=001 asserts its $\overline{\text{BRx}}$ ($\overline{\text{BR1}}$) during reset and for at least two cycles after reset. If no other $\overline{\text{BRx}}$ lines are asserted during these cycles, the DSP with ID=001 drives the memory control sig-

nals to prevent them from glitching. Although it is asserting its $\overline{\text{BRx}}$ and driving the memory control signals during these cycles, this DSP does not perform reads or writes over the bus.

If the DSP with ID=001 is synchronized by the end of the two cycles following reset, it becomes the bus master. If it is not synchronized at this time, it deasserts its $\overline{\text{BRx}}$ ($\overline{\text{BR1}}$) and waits until it is synchronized.



When a DSP has synchronized itself, it sets the BSYN bit in the SYSTAT register.

If one DSP comes out of reset after the others have synchronized and started program execution, that DSP may not be able to synchronize immediately (for example, if it detects more than one $\overline{\text{BRx}}$ line asserted). If the un-synchronized processor tries to execute an instruction with an off-chip read or write, it cannot assert its $\overline{\text{BRx}}$ line to request the bus and execution is delayed until it can synchronize and correctly arbitrate for the bus.

Synchronization cannot occur while $\overline{\text{HBG}}$ is asserted, because bus arbitration is suspended while the bus is controlled by a host. If $\overline{\text{HBR}}$ is asserted immediately after reset and no bus arbitration has taken place, the DSP with ID=001 is considered to be the last bus master.

The DSP with ID=001 maintains correct logic levels on the $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, $\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, PAGE, and $\overline{\text{HBG}}$ signals during reset. Because the “001” processor can be accidently reset by an erroneous write to the soft reset bit (SRST) of the SYSCON register, it behaves in the following manner during reset.

- While it is in reset, the DSP with ID=001 attempts to gain control of the bus by asserting $\overline{\text{BR1}}$.
- While it is in reset, the DSP with ID=001 drives the $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, $\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, $\overline{\text{DMAG1}}$, $\overline{\text{DMAG2}}$, PAGE, and $\overline{\text{HBG}}$ signals only if it determines that it has control of the bus. For the DSP to decide it has

control of the bus, two conditions must be true: 1) $\overline{\text{BRI}}$ was asserted and no other $\overline{\text{BRx}}$ lines were asserted in the previous cycle, and 2) $\overline{\text{HBG}}$ was deasserted in the previous cycle.

Timing differences occur during processor reset ($\overline{\text{RESET}}$) or software reset (SRST bit in SYSCON register = 1) deassertion ($\overline{\text{MS3-0}}$, $\overline{\text{HBG}}$, $\overline{\text{DMAGx}}$, $\overline{\text{RDx}}$, $\overline{\text{WRx}}$, $\overline{\text{CIF}}$, $\overline{\text{PAGE}}$, $\overline{\text{BRST}}$) and threestate (FLAG3-0, LxCLK, LxACK, LxDAT7-0, ACK, REDY, $\overline{\text{PA}}$, TFSx, RFSx, TCLKx, RCLKx, DTx, $\overline{\text{BMS}}$, TDO, $\overline{\text{EMU}}$, DATA). These timings occur asynchronously to CLKIN and may not meet the specifications published in the *ADSP-21535 DSP Microcomputer Data Sheet* Timing Requirements and Switching Characteristics tables. Refer to the *ADSP-21535 DSP Microcomputer Data Sheet* for more information.

The DSP with ID=001 continues to drive the $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, $\overline{\text{MS3-0}}$, $\overline{\text{CIF}}$, $\overline{\text{DMAG1}}$, $\overline{\text{DMAG2}}$, $\overline{\text{PAGE}}$, and $\overline{\text{HBG}}$ signals for two cycles after reset, as long as neither $\overline{\text{HBG}}$ nor any other $\overline{\text{BRx}}$ lines are asserted. At the end of the second cycle it assumes bus mastership (if it is synchronized), and normal bus arbitration begins in the following cycle. If it is not synchronized, it deasserts $\overline{\text{BRI}}$, stops driving the memory control signals and does not arbitrate for the bus until it becomes synchronized.

Although the bus synchronization scheme allows individual processors to be reset, the DSP with ID=001 may fail to drive the memory control signals if it is in reset while any other processors are asserting their $\overline{\text{BRx}}$ line.

If the DSP with ID=001 has asserted $\overline{\text{HBG}}$ while it is in reset, it is synchronized when $\overline{\text{RESET}}$ is deasserted. This lets the host start using the bus while the DSPs are still in reset.

If a host processor attempts to reset the DSP bus master (which is driving the $\overline{\text{HBG}}$ output), the host immediately loses control of the bus.

During reset, the ACK line is pulled high internally by the DSP bus master with a 2 kΩ equivalent resistor.

Booting Another DSP

If the system uses one DSP to boot another DSP over the cluster bus, the master DSP must do the following to communicate to the slave DSP (DSP that boots) through the external port interface:

1. Program the PMODE field in DMAC10 of the booting DSP for no packing.
2. Make 48-bit writes to EPB0 on the booting DSP.

Multiprocessor Direct Writes and Reads

A DSP bus master has the same type of access as a host processor to read or write the internal memory and I/O processor registers of a slave DSP. A DSP bus master or host processor can access the slave by reading and writing to the appropriate address in multiprocessor memory space—this is called a direct read or direct write access. For more information, see "Slave Direct Reads and Writes" on page 7-68.

Each DSP bus slave monitors addresses driven on the external bus and responds to any that fall within its region of multiprocessor memory space. These accesses are invisible to the slave DSP's processor core. They do not degrade internal memory or internal bus performance as seen by the core. This feature lets the processor core continue program execution uninterrupted.

The DSP bus master can directly read and write the slave's I/O processor registers (for example, SYSCON, SYSTAT) to send a vector interrupt or to set up DMA transfers.

The DSP supports 64-bit direct writes through normal word (32-bit) address space. The master can perform a 64-bit DMA to a slave by asserting Normal word addresses in multiprocessor slave space with the stride set to 2. The master can also do 64-bit direct writes by asserting Normal

word addresses in multiprocessor slave space with the LW mnemonic set or with SIMD enabled.

IOP Shadow Registers

In a multiprocessing system, read access to another DSP's PC or MODE2 register provides useful information. The DSP's I/O processor registers include registers that shadow or mirror the program counter (PC), and MODE2 registers. For more information, see "IOP Shadow Registers" on page 7-68.

Instruction Transfers

Multiprocessor instruction transfers to or from internal memory of DSP should use 64-bit transfers for maximum performance (described below). If 48-bit internal transfers are required, one of the slave EPBx FIFOs must be employed, using the packing mode function (PMODE) of the DMA channel.

Maximum throughput is achieved by transferring packed instructions to or from internal memory, using the full 64-bit data bus width synchronously. For more information, see the **Packed** versus **Unpacked** instruction discussion on on page 6-27.

For packed 64-bit direct reads or writes of 48-bit memory the addresses must be translated as during a host processor transfer. For more information, see "Instruction Transfers" on page 7-69.

Direct Writes

When a direct write to a slave DSP occurs, the address, data, and control are latched into a dedicated direct write FIFO. For more information, see "Direct Writes" on page 7-70.

Direct Write Latency

The DSP handles asynchronous and synchronous direct writes differently. This difference influences the latency for the direct writes. For more information, see "Direct Write Latency" on page 7-71.

Direct Reads

When a direct read of a DSP occurs, the address is latched on-chip by the I/O processor at the end of the first CLKIN cycle. ACK is deasserted in the following CLKIN cycle. When the data is available, the I/O processor drives the data and asserts REDY (or ACK). Direct reads cannot be pipelined like direct writes—they only occur one at a time. See "Direct Reads" on page 7-72.

Broadcast Writes

Broadcast writes allow simultaneous transmission of data to all of the DSPs in a multiprocessing system. The master DSP can perform broadcast writes to the same memory location or I/O processor register on all of the slaves. During broadcast writes, the master also writes to itself unless the broadcast is a DMA write. Broadcast writes can be used to implement reflective semaphores in a multiprocessing system. For more information, see "Bus Lock and Semaphores" on page 7-121. Broadcast writes also can simultaneously transfer code or data to multiple processors.

The highest region of multiprocessor memory space, addresses 0x0070 0000 to 0x007F FFFF, is used for broadcast writes. When a write address falls within this region, each DSP slave responds by accepting the access; the master DSP also accepts its own broadcast write. A read cycle generated in the broadcast write region reads the corresponding location in that processor's internal memory and does not assert the processor's $\overline{\text{BRx}}$.

Figure 7-34 shows the timing for a typical broadcast write. In this example, the first broadcast write is accepted by all slaves in cycle 1. This

broadcast write fills the write buffer capacity of one or more of the DSP slaves for less than three CLKIN cycles. The second broadcast write stalls on the bus until write capacity is available in all of the slaves—as indicated by none of the slaves deasserting ACK in cycle 4. The master—having sampled ACK deasserted at the end of cycle 2—pre-charges ACK in cycle 3. The master samples ACK asserted at the end of cycle 4, indicating that all slaves have accepted the second broadcast write.

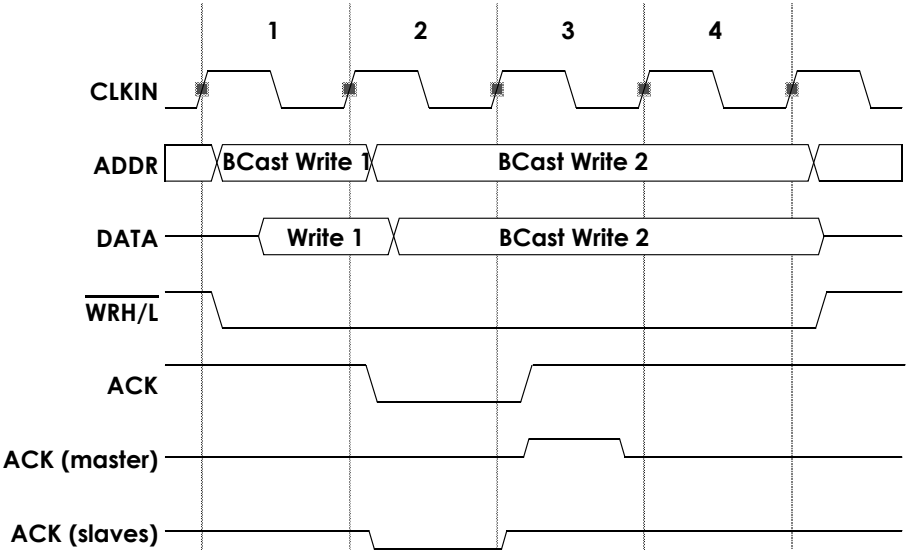


Figure 7-34. Broadcast Write Timing Example

Because the master DSP must wait for a broadcast write to complete on all of the slaves, the acknowledge signal is handled differently to prevent drive conflicts on the ACK line. A wired-OR acknowledge signal is implemented to respond to broadcast writes.

This protocol operates as follows.

1. The DSP does not assert the strobes for broadcast write, unless it samples ACK asserted at the end of the previous cycle.
2. The synchronous broadcast write completes on the bus in the first cycle if ACK is sampled asserted at the end of that cycle.
3. In the first cycle of the broadcast write and in all succeeding odd cycles, a slave DSP deasserts ACK if it is not ready to allow the broadcast write to complete on the bus. If it is ready, it does not drive the ACK line.
4. During all succeeding even cycles in which the broadcast write is not finished, the slave DSPs do not drive ACK. Instead, the master DSP drives (for example, pre-charges) ACK high and must continue the write. (Iterate steps 3 and 4).

In most cases, the ACK signal is high, and the DSP slaves are ready to accept data at the start of the broadcast write—the write completes in one cycle. If the ACK signal is low or one of the slaves is not ready to accept the data, the broadcast write takes a minimum of three cycles.



The DSP with ID=00x enables a keeper latch on the ACK line to prevent the signal from drifting. This eliminates any power consumption caused by the signal drifting to the switching point and improves the robustness of broadcast writes. Multiprocessor systems that use broadcast writes should keep the ACK signal line as free of noise as possible.

Shadow Write FIFO

Because the DSP's internal memory must operate at high speeds, writes to the memory do not go directly into the memory array, but rather to a two-deep FIFO called the shadow write FIFO. The operation of this

FIFO is transparent to program execution. For more information, see "Shadow Write FIFO" on page 7-73.

Data Transfers Through the EPBx Buffers

The DSP bus master can transfer data to and from the slave DSPs through the external port FIFO buffers, EPB0, EPB1, EPB2, and EPB3. Each of these buffers, which are part of the I/O processor register set, is an eight-location FIFO. Both single-word transfers and DMA transfers can be performed through the EPBx buffers. DMA transfers are handled internally by the DSP's I/O processor, but single-word transfers must be handled by the DSP core. For more information, see "Data Transfers Through the EPBx Buffers" on page 7-74.

The DSP supports synchronous burst read transfers from the EPBx FIFOs, or direct read from internal memory, as a slave. Burst write transfers are not supported. Burst reads are supported as contiguous, aligned, 64-bit data transfers up to a maximum length of four 64-bit transfers. The DSP slave increments the address if the burst read access is from internal memory space only. The slave address increment function is only supported for ADDR2-1 (similar to SBSRAMs).

To perform a burst read transfer from an EPBx buffer, the DSP master issues a starting burst address pointing to one of the EPBx buffer addresses in I/O processor control register space. The DSP slave does not increment an EPBx burst read address, and the master DSP limits the burst transfer length to the modulo4 address boundary restriction.

Bus Lock and Semaphores

Semaphores can be used in multiprocessor systems to allow the processors to share resources such as memory or I/O. A semaphore is a flag that can be read and written by any of the processors sharing the resource. The value of the semaphore tells the processor when it can access the resource.

Semaphores are also useful for synchronizing the tasks being performed by different processors in a multiprocessing system.

With the use of its bus lock feature, the DSP has the ability to read and modify a semaphore in a single indivisible operation—a key requirement of multiprocessing systems.

Because both external memory and each DSP's internal memory (and I/O processor registers) are accessible by every other DSP, semaphores can be located almost anywhere. Read-modify-write operations on semaphores can be performed if all of the DSPs obey two simple rules:

- A DSP must not write to a semaphore unless it is the bus master. This is especially important if the semaphore is located in the DSP's own internal memory or I/O processor registers.
- When attempting a read-modify-write operation on a semaphore, the DSP must have bus mastership for the duration of the operation.

Both rules apply when a DSP uses its bus lock feature, which retains its mastership of the bus and prevents the other processors from simultaneously accessing the semaphore.

Bus lock is requested by setting the BUSLK bit in the MODE2 register. When this happens, the DSP initiates the bus arbitration process in the usual fashion, by asserting its $\overline{\text{BRx}}$ line. When it becomes bus master, it locks the bus (for example, retains bus mastership) by keeping its $\overline{\text{BRx}}$ line asserted even when it is not performing an external read or write. Host Bus Request ($\overline{\text{HBR}}$) is also ignored during a bus lock. When the $\overline{\text{BUSLK}}$ bit is cleared, the DSP gives up the bus by deasserting its $\overline{\text{BRx}}$ line.

While the BUSLK bit is set, the DSP can determine if it has acquired bus mastership by executing a conditional instruction with the Bus Master (Bm) or Not Bus Master (Not Bm) condition codes, for example:

```
IF NOT BM JUMP(PC,0); /* wait for bus mastership */
```

If it has become the bus master, the DSP can proceed with the external read or write. If not, it can clear its BUSLK bit and try again later.

A read-modify-write operation is accomplished with the following steps.

1. Request bus lock by setting the BUSLK bit in MODE2.
2. Wait for bus mastership to be acquired.
3. Wait until Direct Write Pending (DWPD) is zero.
4. Read the semaphore, test it, then write to it.

Locking the bus prevents other processors from writing to the semaphore while the read-modify-write is occurring. After bus mastership is acquired, the Direct Write Pending (DWPD) bit's status in SYSTAT must be checked to ensure that a semaphore write by another processor is not pending.



If the semaphore is reflective, located in the DSP's internal memory or an I/O processor register, the processor must write to it only when it has bus lock.

Bus lock can be used in combination with broadcast writes to implement reflective semaphores in a multiprocessing system. The reflective semaphore should be located at the same address in internal memory or I/O processor register of each DSP. To check the semaphore, each DSP simply reads from its own internal memory. To modify the semaphore, a DSP requests bus lock and then performs a broadcast write to the semaphore address on every DSP, including itself. Before modifying the semaphore, the DSP must re-check it to verify that another processor has not changed it. With reflective semaphores, the external bus is used only for updating the semaphore, not for reading it. This technique reduces bus traffic.

Interprocessor Messages and Vector Interrupts

The DSP bus master can communicate with slave DSPs by writing messages to their I/O processor registers. The MSGR0-7 registers are

general-purpose registers which can be used for convenient message passing between DSPs. They are also useful for semaphores and resource sharing between multiple DSPs. The MSGRx and VIRPT registers can be used for message passing in the following ways.

- Message Passing. The master DSP can communicate with a slave DSP by writing and/or reading any of the 8 message registers, MSGR0-MSGR7, on the slave.
- Vector Interrupts. The master DSP can issue a vector interrupt to a slave by writing the address of an interrupt service routine to the slave's VIRPT register. This causes an immediate high-priority interrupt on the slave which, when serviced, causes it to branch to the specified service routine.

The MSGRx and VIRPT registers also support the host processor interface. Because these registers may be shared resources within a single DSP, conflicts may occur—system software must prevent this. For further discussion of I/O processor register access conflicts, see "I/O Processor Registers" on page A-33

Message Passing (MSGRx)

There are three methods by which the DSP bus master can communicate with a slave through the MSGRx message registers: 1) vector-interrupt driven, 2) register handshake, and 3) register write-back. These techniques are the same as for a host processor. For more information, see "Message Passing (MSGRx)" on page 7-82.

Vector Interrupts (VIRPT)

Vector interrupts are used for interprocessor commands between two DSPs or between a host and the DSP. When the external processor writes an address to the DSP's VIRPT register, a vector interrupt is caused. Vector interrupts operate the same for host and multiprocessor systems. For more information, see "Host Vector Interrupts (VIRPT)" on page 7-82.

Multiprocessor Interface Status

The SYSTAT register provides status information for host and multiprocessor systems. Table 7-14 shows the status bits in this register. For more information on the SYSTAT register, see Table A-20 on page A-51.

Table 7-14. SYSTAT System Status Register

Bit #	Name	Definition
0	HSTM	Host Mastership – Indicates whether the host processor has been granted control of the bus. 1=Host is bus master 0=Host is not bus master
1	BSYN	Bus Synchronization – Indicates when the DSP's bus arbitration logic is synchronized after reset. For more information, see "Bus Synchronization After Reset" on page 7-113. 1=Bus arbitration logic is synchronized 0=Bus arbitration logic is not synchronized
[3:2]		reserved
[6:4]	CRBM	Current Bus Master – Indicates the ID of the DSP that is the current bus master. If CRBM is equal to the ID of this DSP then it is the current bus master. CRBM is only valid for ID 2-0 > 0 (greater than zero). When ID 2-0 =000, CRBM is always 1.
7		reserved
[10:8]	IDC	ID Code – Indicates the ID 2-0 inputs of this DSP.
11		reserved
12	DWPD	Direct Write Pending – Indicates when a direct write to the DSP's internal memory is pending. The DWPD bit is cleared when the direct write has been completed. (Direct writes may be delayed for several cycles if DMA chaining is underway or if higher priority DMA requests occur. Maximum delay is 12 cycles.) 1=Direct write pending 0=No direct write pending

Table 7-14. SYSTAT System Status Register (Cont'd)

Bit #	Name	Definition
13	VIPD	Vector Interrupt Pending – Indicates that a pending vector interrupt has not yet been serviced. The VIPD bit is set when the VIRPT register is written to and is cleared upon return from the interrupt service routine. The host processor (or other DSP) that issued the vector interrupt should monitor this bit to determine when the service routine has been completed (and when a new vector interrupt may be issued). 1=Vector interrupt pending 0=No vector interrupt pending
[15:14]	HPS	Host Packing Status – Indicates when host word packing is completed or, if not, what stage of the packing/unpacking process is taking place. 00=process complete 01=First stage of process 10=Second stage of process
[19:16]	CRAT	Processor Core Clock (CCLK)-to-CLKIN clock ratio, as determined by the CLK_CFG0-3 inputs
[31:20]		reserved

8 LINK PORTS

This chapter describes the ADSP-21160 DSP's link ports. The DSP has six 8-bit wide link ports, which can connect to other DSPs' or peripherals' link ports.

Overview

The six ADSP-21160 DSP's bidirectional link ports have eight data lines, an acknowledge line, and a clock line. Link ports can operate at frequencies up to the same speed as the DSP's internal clock, letting each port transfer up to 8 bits of data per internal clock cycle.

Link ports also:

- Operate independently and simultaneously.
- Pack data into 32- or 48-bit words; this data can be directly read by the DSP or DMA-transferred to or from on-chip memory.
- Are accessible by the external host processor, using direct reads and writes.
- Have double-buffered transmit and receive data registers.

- Include programmable clock/acknowledge controls for link port transfers. Each link port has its own dedicated DMA channel.
- Provide high-speed, point-to-point data transfers to other DSP processors, allowing differing types of interconnections between multiple DSPs, including 1-, 2- and 3-dimensional arrays.

ADSP-21160 DSP's link ports are logically (but not electrically) compatible with previous SHARC DSP (ADSP-2106x DSPs) link ports. For more information, see "Link Data Path (and Compatibility) Modes" on page 8-7.

Table 8-1 on page 8-2 lists the pins associated with each link port. Each link port consists of eight data lines (LxDAT7-0), a link clock line (LxCLK), and a link acknowledge line (LxACK). The LxCLK line allows asynchronous data transfers and the LxACK line provides handshaking. When configured as a transmitter, the port drives both the data and LxCLK lines. When configured as a receiver, the port drives the LxACK line.

Figure 8-1 shows link port connections.

Table 8-1. Link Port Pins

Link Port Pin(s)	Link Port Function
LxDAT7-0	Link Port x Data
LxCLK	Link Port x Clock
LxACK	Link Port x Acknowledge
"x" denotes the link port number, 0-5	

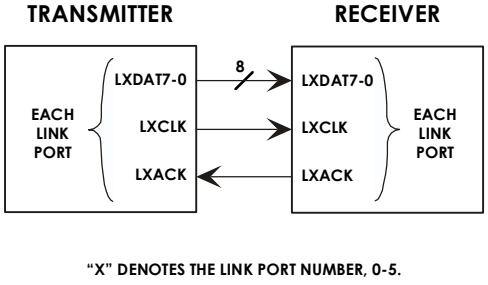


Figure 8-1. Link Port Pin Connections

Link Port To Link Buffer Assignment

There are six buffers, LBUF0-5, that buffer the data flow through the link ports. These buffers are independent of the link ports and may be connected to any of the six link ports. The link ports receive and transmit data on their LxDAT7-0 data pins. Any of the six link buffers may be assigned to handle data for a particular link port. The data in the link buffers can be accessed with DMA or processor core control.



"Link Port x" does not necessarily connect to "Link Buffer x."

Figure 8-2 shows a block diagram of the link ports and link buffers.

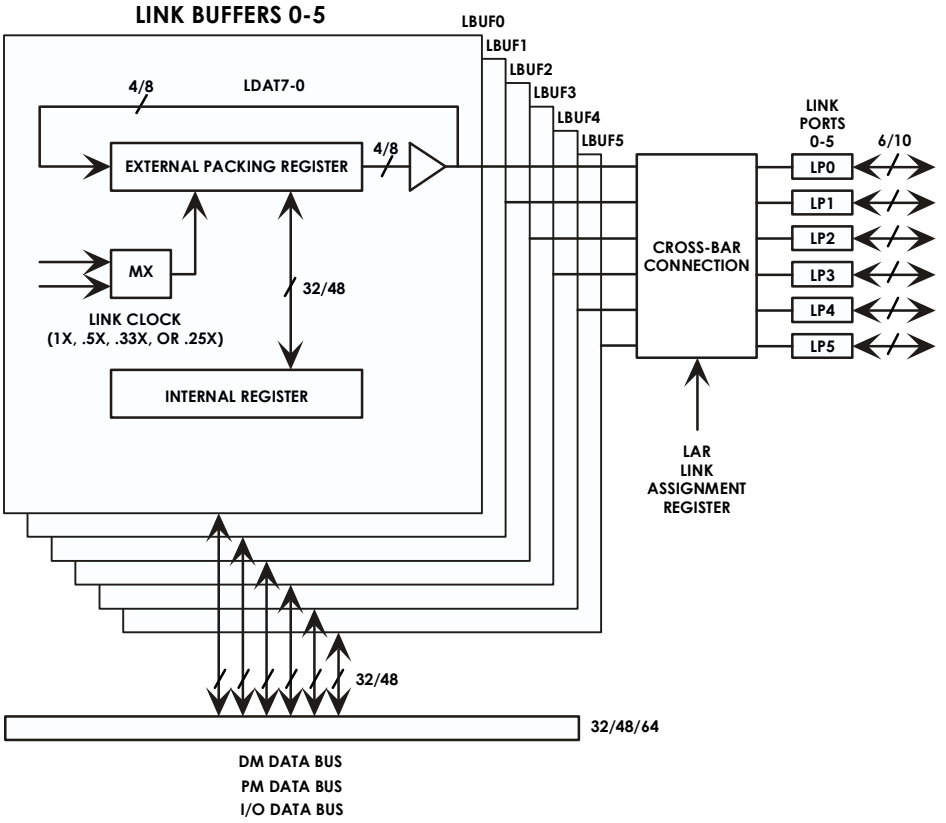


Figure 8-2. Link Ports and Buffers

The Link Assignment Register (LAR) assigns the link buffer-to-port connections. Memory-to-memory transfers may be accomplished by assigning the same link port to two buffers, setting up a loopback mode.

For details on the LAR register, see "Link Port Assignment Register (LAR)" on page A-67.

Assigning more than two buffers to one port will disable the port.

Link Port DMA Channels

DMA channels 4-9 support buffers 0-5. The buffer channel pairings are listed in Table 8-2.

Table 8-2. DMA Channel/Link Buffer Pairing

DMA Channel #	Link Buffer Supported
DMA Channel 4	Link Buffer 0
DMA Channel 5	Link Buffer 1
DMA Channel 6	Link Buffer 2
DMA Channel 7	Link Buffer 3
DMA Channel 8	Link Buffer 4
DMA Channel 9	Link Buffer 5

For more information, see “Setting I/O Processor—LPort Modes” on page 6-41.

Link Port Booting

Systems may boot the DSP through a link port. For more information, see “Bootloading Through The Link Port” on page 6-88.

Setting Link Port Modes

The SYSCON, LCOM, LAR, and LCTLx registers control the link ports operating modes for the I/O processor. Table A-17 on page A-45 lists all the bits in SYSCON, Table A-24 on page A-65 lists all the bits in LCOM, Table A-25 on page A-67 lists all the bits in LAR, and Table A-22 on page A-63 lists all the bits in LCTLx.

The following bits control link port modes. Some other bits in the SYSCON, LCOM, LAR, and LCTLx registers setup DMA and I/O processor related link

port features. For information on these features, see For more information, see “Setting I/O Processor—LPort Modes” on page 6-41..

- Link Buffer Mesh Multiprocessing. LCOM Bit 20 (LMSP) This bit enables (if set, =1) or disable (if cleared, =0) mesh multiprocessing.
- Link Path (Mesh Multiprocessing) Delay. LCOM Bit 22-21 (LPATHD) These bits apply change over delays when changing to the next LPATH register as follows: 00=no additional delay, 01=1 additional delay, 10=2 additional delays, 11=3 additional delays.

  Link Port Clock Divisor. LCTL0 Bits 6-5, 16-15, and 26-25 and LCTL1 Bits 6-5, 16-15, and 26-25 (LxCLKD) These bits select the transfer clock divisor for the corresponding link buffer (LBUFx). The transfer clock equals the DSP’s internal clock (CCLK) divided by LxCLKD, where LxCLKD is: 01=1, 10=2, 11=3, or 00=4.
- Link Port pull-down Resistor Disable/Enable. LCTL0 Bits 8, 18, and 28 and LCTL1 Bits 8, 18, and 28 (LxPDRDE) This bit disables (if set, =1) or enables (if cleared, =0) the internal pull-down resistors

on the LxCLK, LxACK, and LxDAT7-0 pins of the corresponding link port; this bit applies to the port, which is not necessarily the port assigned to the corresponding link buffer (LBUFx).



If multiple link ports are bussed together and you have the Link Port pull-down Resistor enabled on all the processors, you will heavily load the line. Ensure that you have only one DSP enabling this functionality.

Link Port Data Path Width. LCTL0 Bits 9, 19, and 29 and LCTL1 Bits 9, 19, and 29 (LxDPWID) This bit selects the link port data path width (8-bit if set, =1) (4-bit if cleared, =0) for the corresponding link buffer (LBUFx).



The DSP's internal clock (CCLK) is the CLKIN frequency multiplied by a clock ratio (CLK_CFG3-0). For more information, see the clock ratio discussion on page 11-8.



When link buffers are enabled or disabled, the I/O processor may generate unwanted interrupt service requests if Link Service Request (LSRQ) interrupts are unmasked. To avoid unwanted interrupts, programs should mask the LSRQ interrupts while enabling or disabling link buffers. For more information, see "Using Link Port Interrupts" on page 8-12.

Link Data Path (and Compatibility) Modes

The link ports can transmit and received data using all 8 of the link port's data pins (LxDAT7-0) or the 4 lower data pins (LxDAT3-0). The LxDPWID bit in the LCTLx register selects the link port data path width (8-bit if set, =1) (4-bit if cleared, =0).



When LxDPWID is cleared (4-bit data path), the ADSP-21535 DSP can be connected to link ports of previous SHARC DSPs (ADSP-2106x DSP family). The link port receiver must run at the same speed or faster than the transmitter. Connecting to an

ADSP-2106x DSP may require that the ADSP-21160 DSP be configured for 1/2 core clock rate operation. For more information, see "Using Link Port Handshake Signals" on page 8-8.

Using Link Port Handshake Signals

The LxCLK and LxACK pins of each link port allow handshaking for asynchronous data communication between DSPs. Other devices that follow the same protocol may also communicate with these link ports. The DSP link ports are backward compatible with the SHARC link ports for basic transfers, including LSRQ functions.

A link-port-transmitted word consists of 4 bytes (for a 32-bit word) or 6 bytes (for a 48-bit word). The transmitter asserts the clock (LxCLK) high with each new byte of data. The falling edge of LxCLK is used by the receiver to latch the byte. The receiver asserts LxACK when it is ready to accept another word in the buffer. The transmitter samples LxACK at the beginning of each word transmission (that is, after every 4 or 6 bytes). If LxACK is deasserted at that time, the transmitter does not transmit the new word. For more information, see Figure 8-3 on page 8-9.

The transmitter leaves LxCLK high and continues to drive the first byte if LxACK is deasserted. When LxACK is eventually asserted again, LxCLK goes low and begins transmission of the next word. If the transmit buffer is

empty, LxCLK remains low until the buffer is refilled, regardless of the state of LxACK.

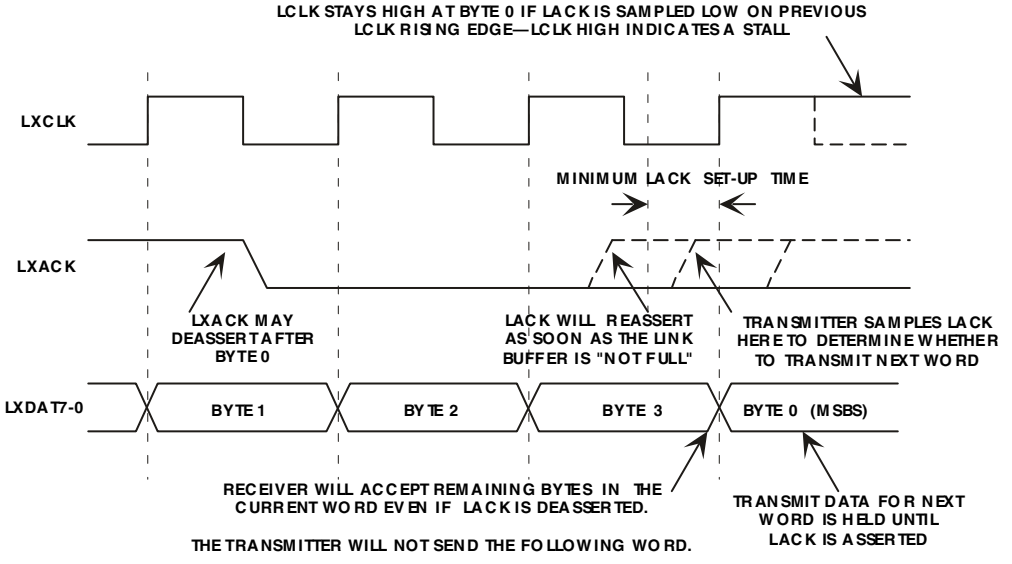

LCLK STAYS HIGH AT BYTE 0 IF LACK IS SAMPLED LOW ON PREVIOUS LCLK RISING EDGE—LCLK HIGH INDICATES A STALL
LXCLK
MINIMUM LACK SET-UP TIME
LXACK
LXACK MAY DEASSERT AFTER BYTE 0
LACK WILL REASSERT AS SOON AS THE LINK BUFFER IS "NOT FULL"
TRANSMITTER SAMPLES LACK HERE TO DETERMINE WHETHER TO TRANSMIT NEXT WORD
LXDAT7-0
BYTE 1
BYTE 2
BYTE 3
BYTE 0 (MSBS)
RECEIVER WILL ACCEPT REMAINING BYTES IN THE CURRENT WORD EVEN IF LACK IS DEASSERTED.
THE TRANSMITTER WILL NOT SEND THE FOLLOWING WORD.
TRANSMIT DATA FOR NEXT WORD IS HELD UNTIL LACK IS ASSERTED

Figure 8-3. Link Port Handshake Timing (32-Bit Mode)

The receive buffer may fill if a higher priority DMA, core I/O processor register access, direct read, direct write or chain loading operation is occurring. LxACK may deassert when it anticipates the buffer may fill. LxACK is reasserted by the receiver as soon as the internal DMA grant signal has occurred, freeing a buffer location.

Data is latched in the receive buffer on the falling edge of LxCLK. The receive operation is purely asynchronous and can occur at any frequency up to the processor clock frequency. When a link port is not enabled, LxDAT7-0, LxCLK and LxACK are three-stated. When a link port is enabled to transmit, the data pins are driven with whatever data is in the output buffer, LxCLK is driven high and LxACK is three-stated. When a port is

enabled to receive, the data pins and LxCLK are three-stated and LxACK is driven high.

To allow a transmitter and a receiver to be enabled (assigned and link buffer enabled) at different times, LxACK, LxCLK, and LxDAT7-0 may be held low with the 50KΩ internal pull-down resistors if LxPDRDE is cleared when the Link Port is disabled. If the transmitter is enabled before the receiver, LxACK is low and the transmission is held off. If the receiver is enabled before the transmitter, LxCLK is held low by the pull-down and the receiver is held off. If many link ports are bused together, the systems may need to enable only one of the internal resistors to pull down each bused pin, so the bused lines are not pulled down too strongly or too heavily loaded.



LxACK, LxCLK, and LxDAT7-0 should not be left unconnected unless external pull-down resistors are used.

Using Link Buffers

Each link buffer consists of an external and an internal 48-bit register. For more information, see Figure 8-2 on page 8-4. When transmitting, the internal register is used to accept core data or DMA data from internal memory. When receiving, the external register performs the packing and unpacking for the link port, most significant nibble or byte first. These two registers form a two-stage FIFO for the LBUFx buffer. Two writes (32- or 48-bit) can occur to the register by the DMA or the core, before it signals a full condition. As each word is unpacked and transmitted, the next location in the FIFO becomes available and a new DMA request is made. If the register becomes empty, the LxCLK signal is de-asserted. When transmitting, only the number of words written are transmitted.

Full/empty status for the link buffer FIFOs is given by the LxSTAT bits of the LCOM register. This status is cleared for a link buffer when its LxEN enable bit is cleared in the LCTLx register.

During receiving, the external buffer is used to pack the receive link port data (most significant nibble or byte first) and pass it to the internal register before DMA-transferring it to internal memory. This buffer is a two-deep FIFO. If the DSP's DMA controller does not service it before both locations are filled, the LxACK signal is de-asserted.

The link buffer width may be selected to be either 32 or 48 bits. This selection is made individually for each buffer with the LxEXT bits in the LCTLx register. For 40-bit extended precision data or 48-bit instruction transfers, the width must be set to 48 bits.

Core Processor Access To Link Buffers

In applications where the latency of link port DMA transfers to and from internal memory is too long, or where a process is continuous and has no block boundaries, the DSP processor core may read or write link buffers directly using the full or empty status bit of the link buffer to automatically pace the operation. The full or empty status of a particular LBUFx buffer can be determined by reading the LCOM control/status register. DMA should be disabled when using this capability (LxDEN=0).

If a read is attempted from an empty receive buffer, the core stalls (hangs) until the link port completes reception of a word. If a write is attempted to a full transmit buffer, the core stalls until the external device accepts the complete word. Up to four words (2 in the receiver and 2 in the transmitter) may be sent without a stall before the receiver core or DMA must read a link buffer register.



To support debugging buffer transfers, the DSP has a Buffer Hang Disable (BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition.

For more information, see the BHD discussion on page -18.

Host Processor Access To Link Buffers

The link buffers can also be accessed by the external host processor, using direct reads and writes. When the host reads or writes to these buffers, the word width is determined only by the host packing mode, as selected by the HPM bits in the SYSCON register.

Using Link Port DMA

DMA channels 4-9 support link buffers 0-5. A maskable interrupt is generated when the DMA block transfer has completed. For more information on link port interrupts, see "Using Link Port Interrupts" on page 8-12. For more information on link port DMA, see "Link Port DMA" on page 6-81.



Unlike previous SHARC DSPs, there are no shared DMA channels on the ADSP-2153x DSP. Each link port buffer has its own dedicated DMA channel.

In chained DMA operations, the DSP automatically sets up another DMA transfer when the current DMA operation completes. The chain pointer register (CPx) is used to point to the next set of buffer parameters stored in memory. The DSP's DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For information on setting up DMA chaining, see "Chaining DMA Processes" on page 6-70.

Using Link Port Interrupts

Three types of interrupts are dedicated to the link ports:

The I/O processor generates a DMA channel interrupt when a DMA block transfer through the link port with DMA enabled (LxDEN=1) finishes.

- The I/O processor generates a DMA channel interrupt when DMA for the link buffer channel is disabled (LxDEN=0) and the buffer is not full (for transmit) or the buffer is not empty (for receive).
- The I/O processor generates a Link Services Request (LSRQ) interrupt when an external source accesses a disabled link port—unassigned link port or assigned port with buffer disabled.

Registers control link port interrupt latching and masking. The LIRPTL register is the individual link port interrupt latch/mask register, and the IRPTL and IMASK registers control global link port DMA interrupt latching and masking. For more information, see "Link Port Interrupt Register (LIRPTL)" on page A-24, "Interrupt Latch Register (IRPTL)" on page A-18, and "Interrupt Mask Register (IMASK)" on page A-22.

Link Port Interrupts With DMA Enabled

A link port interrupt is generated when the DMA operation is done—when the block transfer has completed and the DMA count register is zero.

One way programs can use this interrupt is to send additional control information at the end of a block transfer. Because the receive DMA buffer is empty when the DMA block has completed, the external bus master can send up to two additional words to the slave DSP's buffer, which has space for the two words. The slave's same interrupt vector associated with the completion of the Link Port DMA could then read the buffer and use these control words to determine the next course of action.

Link Port Interrupts With DMA Disabled

If DMA is disabled for a link port buffer, then the buffer may be written or read by the DSP core as a memory-mapped I/O processor register.

If the DMA is disabled but the associated link buffer is enabled, then a maskable interrupt is generated whenever a receive buffer is not empty or when a transmit buffer is not full. This interrupt is the same interrupt vector associated with the completion of the DMA block transfers.

The interrupt latch bit in LIRPTL may be unmasked by the corresponding mask bit in the same register. When initially enabling the mask bit, the corresponding latch bit in LIRPTL should be cleared first to clear out any request that may have been inadvertently latched.

The interrupt service routine should test the buffer status after each read or write to check when the buffer is empty or full, in order to determine when it should return from interrupt. This will reduce the number of interrupts it must service.

Link Port Service Request Interrupts (LSRQ)

Link port service requests let a disabled (unassigned or assigned with buffer disabled) link port cause an interrupt when an external access is attempted. The transmit and receive request status bits of the LSRQ register let a DSP determine if another DSP is attempting to send or receive data through a particular link port. This lets two processors communicate without prior knowledge of the transfer direction, link port number, or exactly when the transfer is to occur.

When LxACK or LxCLK is asserted externally, a link service request (LSR) is generated in a disabled (unassigned or assigned with buffer disabled) link port. LSRs are not generated for a link port that is disabled by loopback mode. Each LSR is gated by mask bits before being latched in the LSRQ register. The six possible receive LSRs and the six possible transmit LSRs are gated by mask bits and then OR'ed together to generate the link ser-

vice request interrupt. The LSRQ interrupt request may be masked by the LSRQI mask bit of the IMASK register. When the mask bit is set, the interrupt is allowed to pass into the interrupt priority encoder. A diagram of this logic appears in Figure 8-4 on page 8-15.

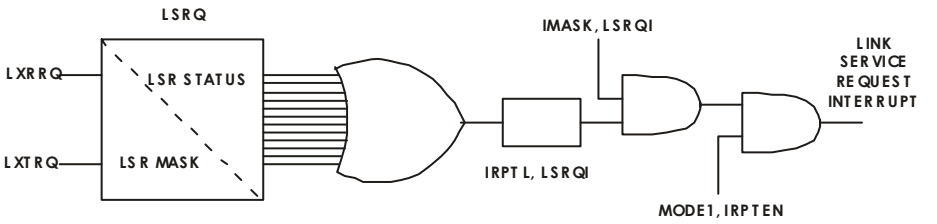


Figure 8-4. Logic For Link Port Interrupts

In Figure 8-4, the LxTRQ and LxRRQ inputs stand for status bits in the LSRQ register. For transmit requests, LxTRQ=1 indicates the following status: LxACK=1, LxTM=1, and LxEN=0. For receive requests, LxRRQ=1 indicates the following status: LxCLK=1, LxRM=1, and LxEN=0.

The interrupt routine must read the LSRQ register to determine which link port to service and whether it is a transmit or receive request. LSR interrupts have a latency of two cycles. Note that the link service request interrupt is different from the link receive and transmit interrupt—this is also true in IMASK.

The 32-bit LSRQ register holds the masked link status of each link port and the corresponding interrupt mask bits. The link service request status of the port is set whenever the port is not enabled and one of LxACK or LxCLK is asserted high. The LSRQ status bits are read-only. Table A-26 on page A-68 shows the individual bits of the LSRQ register.

To determine which link port to service, programs can transfer LSRQ to a register Rx (in the register file) then use the leading 0s detect instruction:

```
Rn=LEFTZ Rx
```

Here, Rn indicates which link port is active in order of priority.

If link service requests are in use, they should be masked out when the assigned link buffers are being enabled, disabled, or when the link port is being unassigned in LAR, otherwise spurious service requests may be generated.

This need for masking is due to a delay before LxCLK or LxACK (if already asserted) signals are pulled (if pull-downs enabled) or driven externally (if pull-downs disabled) below logic threshold. During this delay, these signals are sampled asserted and generate an LSRQ.



To avoid the possibility of spurious interrupts, programs should mask the LSRQ interrupt or the appropriate request bit in the LSRQ register and allow an appropriate delay before unmasking. Alternatively, programs can mask the LSRQ interrupt and poll the appropriate request status bit until it is cleared and then unmask the interrupt.

Detecting Errors On Link Transmissions

Transmission errors on the link ports may be detected by reading the LRERRx bits in the LCOM register. These bits reflects the status of each nibble or byte counter. The LRERRx bit is zero if the pack counter of the corresponding link buffer is zero—a multiple of 8 or 12 nibbles or bytes have been received. If LRERR is high when a transmission has completed, then an error occurred during transmission.



The DMA word count provides an exact count of the number of words to be transferred.

To allow checking of this status, the transmitter and receiver should follow a protocol such as the following:

- Transmitter Protocol—To make use of the LRERRx status, one additional dummy word should always be transmitted at the end of a block transmission. The transmitter must then deselect the link port and re-enable as a receiver to allow the receiver to send an appropriate message back to the transmitter.
- Receiver Protocol—When the receiver has received the data block, indicated by a the same interrupt vector associated with the completion of the Link Port DMA, it checks that it has received an additional word in the link buffer and then reads the LRERR bit. The receiver may then clear the link buffer (LxEN=0) and transmit the appropriate message back to the transmitter on the same, or a different, link port.

Using Token Passing With Link Ports

Two DSPs that communicate using a link port need to know which of them is currently the transmitter and which is the receiver, otherwise they might both try to transmit at the same time. Token passing is a protocol that can help the DSPs alternate control. Figure 8-5 on page 8-18 shows a Token Passing Flow Chart describing a software protocol for token passing through the link ports. A good example of this protocol is available in the Engineer-to-Engineer Note EE-16: Using Token Passing to Control SHARC Link Port Bi-directional Communication on the Analog Devices website.

In token passing, the token is a software flag that passes between the processors. At reset, the token (flag) is set to reside in the link port of one device, making it the master and the transmitter. When a receiver link port (slave) wants to become the master, it may assert its LxACK line (request data) to get the master's attention. The master knows, through

software protocol, whether it is supposed to respond with actual data or whether it is being asked for the token.

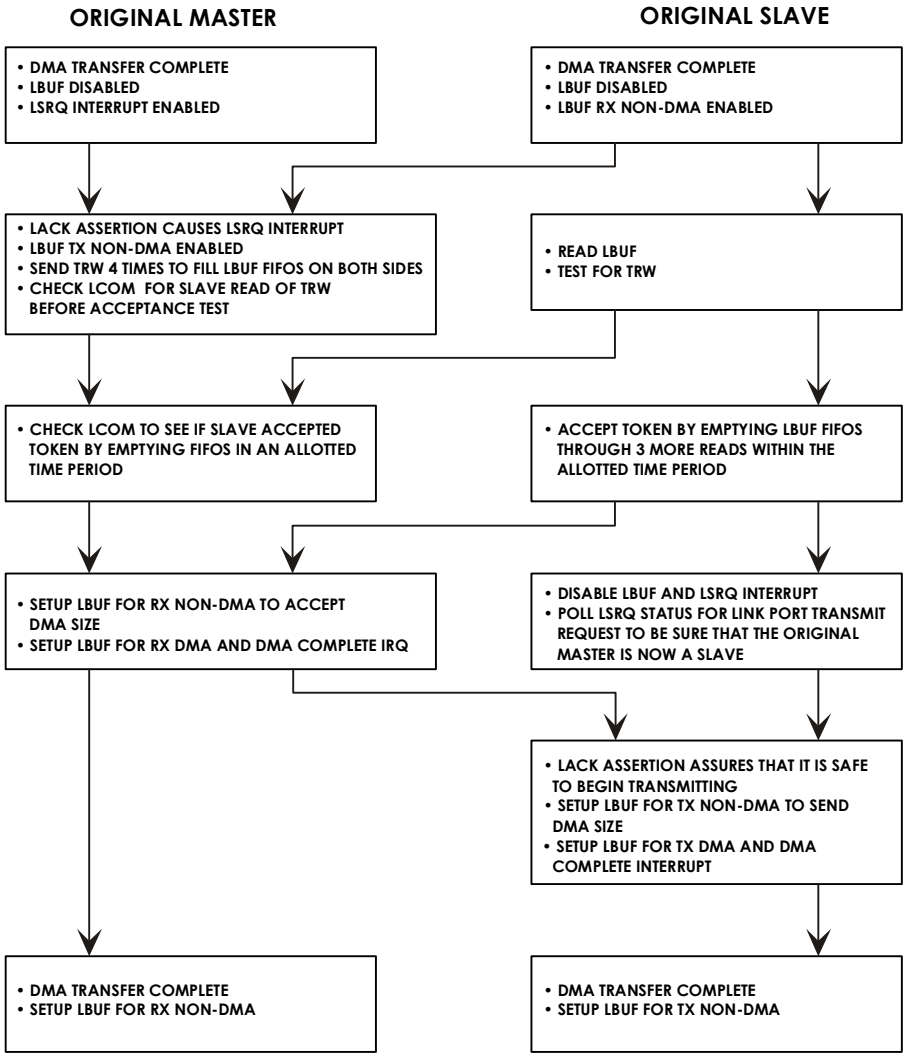

ORIGINAL MASTER
ORIGINAL SLAVE
• DMA TRANSFER COMPLETE
• LBUF DISABLED
• LSRQ INTERRUPT ENABLED
• DMA TRANSFER COMPLETE
• LBUF DISABLED
• LBUF RX NON-DMA ENABLED
• LACK ASSERTION CAUSES LSRQ INTERRUPT
• LBUF TX NON-DMA ENABLED
• SEND TRW 4 TIMES TO FILL LBUF FIFOS ON BOTH SIDES
• CHECK LCOM FOR SLAVE READ OF TRW BEFORE ACCEPTANCE TEST
• READ LBUF
• TEST FOR TRW
• CHECK LCOM TO SEE IF SLAVE ACCEPTED TOKEN BY EMPTYING FIFOS IN AN ALLOTTED TIME PERIOD
• ACCEPT TOKEN BY EMPTYING LBUF FIFOS THROUGH 3 MORE READS WITHIN THE ALLOTTED TIME PERIOD
• SETUP LBUF FOR RX NON-DMA TO ACCEPT DMA SIZE
• SETUP LBUF FOR RX DMA AND DMA COMPLETE IRQ
• DISABLE LBUF AND LSRQ INTERRUPT
• POLL LSRQ STATUS FOR LINK PORT TRANSMIT REQUEST TO BE SURE THAT THE ORIGINAL MASTER IS NOW A SLAVE
• LACK ASSERTION ASSURES THAT IT IS SAFE TO BEGIN TRANSMITTING
• SETUP LBUF FOR TX NON-DMA TO SEND DMA SIZE
• SETUP LBUF FOR TX DMA AND DMA COMPLETE INTERRUPT
• DMA TRANSFER COMPLETE
• SETUP LBUF FOR RX NON-DMA
• DMA TRANSFER COMPLETE
• SETUP LBUF FOR TX NON-DMA

Figure 8-5. Token Passing Flow Chart

The token release word can be any user-defined value. Since both the transmitter and receiver are expecting a code word, this need not be exclusive of normal data transmission.

If the master wishes to give up the token, it may send back a user-defined token release word and thereafter clear its token flag. Simultaneously, the slave examines the data sent back and if it is the token release word, the slave will set its token, and can thereafter transmit. If the received data is not the token release word, then the slave must assume the master was beginning a new transmission.

Through software protocol, the master can also ask to receive data by sending the token release word without the LxACK (data request) going low first. Figure 8-5 shows a flow chart of the example code's protocol.

To use the example, the example code is to be loaded on both the original master and the original slave. The code is ID intelligent for multiprocessor systems: ID1 is the original master (transmitter) and ID2 is original slave (receiver). The master transmits a buffer via DMA through LPORT0 using LBUF3 and the slave receives through LPORT0 using LBUF2. The slave then requests the token by generating an LSRQ interrupt in the disabled link port of the master (LPORT0). The master responds by sending the token release word and waiting to see if it is accepted. The slave checks to see that it is the token release word and accepts the token by emptying the master's link buffer FIFO within a predetermined amount of time.

If the token is accepted the slave becomes the master and transmits a buffer of data to the new slave. If the token is rejected, the master transmits a second buffer. When complete, the original master will finish by setting up LBUF2 to receive without DMA, and the original slave sets up LBUF3 to transmit without DMA.

The following is a list of the major areas of concern when a program implements a software protocol scheme for token passing:

- The program must make sure that both link buffers are not enabled to transmit at the same time. In the event that this is allowed, data may be transmitted and lost due to the fact that neither link port is driving LxACK.
- In the example, the LSRQ register status bits are polled to ensure that the master becomes the slave before the slave becomes the master, avoiding the two transmitter conflict.
- The program must make sure that the link interrupt selection matches the application. If a status detection scheme using the status bits of the LSRQ register is to be used, it is important to note the following:

  If a link port that is configured to receive is disabled while LxACK is asserted, there is an RC delay before the 50KΩ pull-down resistor on LxACK (if enabled) can pull the value below logic threshold.

  If a link port that is configured to transmit is disabled while LxCLK is asserted, there is an RC delay before the 50KΩ pull-down resistor on LxCLK (if enabled) can pull the value below logic threshold.

  If the appropriate request status bit is unmasked in the LSRQ register (in this instance), then an LSR is latched and the LSRQ interrupt may be serviced, even though unintended, if enabled.
- The program must make sure that synchronization is not disrupted by unrelated influences at critical sections where timing control loops are used to synchronize parallel code execution. Disabling of nested interrupts is one techniques to control this.

Designing Link Port Systems

The DSPs link ports support I/O with peripherals and other DSP link ports. While link ports require few connections, there are a number of design issues that systems using these ports must accommodate.

Terminations For Link Transmission Lines

The link ports are designed to allow long distance connections to be made between the driver and the receiver. This is possible because the links are self-synchronizing—the clock and data are transmitted together. Only relative delay, not absolute delay between clock and data is of importance.

In addition, the LACK signal inhibits transmission of the next word, not of the current nibble or byte. For example, the current word is always allowed to complete transmission. This allows delays of 3 to 5 cycles for the LxACK signal to reach the transmitter.

The links are designed to drive transmission lines with characteristic impedances of 50Ω or greater. A higher transmission line impedance reduces the on-chip effect of driver impedance variations, for distances longer than about six inches. It is recommended that an external series termination resistor be used at each link port pin to absorb reflections from the open circuit at the destination. The external resistor should be selected such that its value (plus the internal resistance of the driver) be equal to the characteristic impedance of the transmission line.



For example, a system with a typical internal drive resistance of 10Ω and a characteristic impedance of 50Ω should use a link port pin resistor of 40Ω.

Peripheral I/O Using Link Ports

The example shown in Figure 8-6 on page 8-23 shows how a multiprocessing system can use link ports to connect to local memories and I/O

devices. An ASIC implements the interface between the link port and DRAM or an I/O device. This minimal hardware solution frees the DSP's external bus for other shared-bus communication. The DRAM and ASIC may be implemented on a single 10-pin SIMM module.

Accesses to the DRAM over a link is most efficient under DMA control. The ASIC receives DMA control information from the link port and sets up the access to the DRAM. It unpacks 16-bit data words from the DRAM or packs 8-bit bytes from the link. At the end of the DMA transfer, an interrupt lets the DSP send new control information to the ASIC. The ASIC always reverts to receive mode at the end of a transfer. The LxACK signal is deasserted by the ASIC whenever a page change, memory refresh cycle, or any other access to the DRAM occurs.

Memory modules may be shared by multiple DSPs when the link port is bused. Each link port supports 100 Mbyte per second access throughput for either instructions or data. The ASIC is responsible for generating the clock when transmitting to the DSP. The ASIC is also responsible for generating sequential DMA addresses based on a start address and word count.

Data Flow Multiprocessing With Link Ports

Figure 8-7 on page 8-24 shows examples of different link port communications schemes.

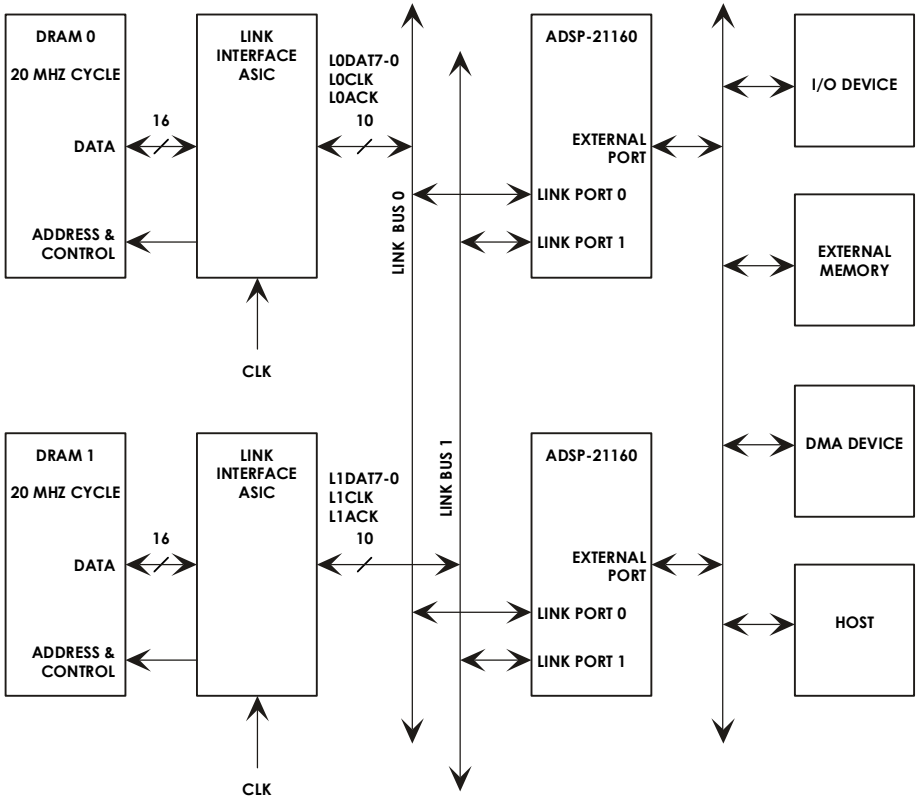


Figure 8-6. Local DRAM With Link Ports

For more information on the multiprocessor interface, see "Multiprocessing System Architectures" on page 7-97.

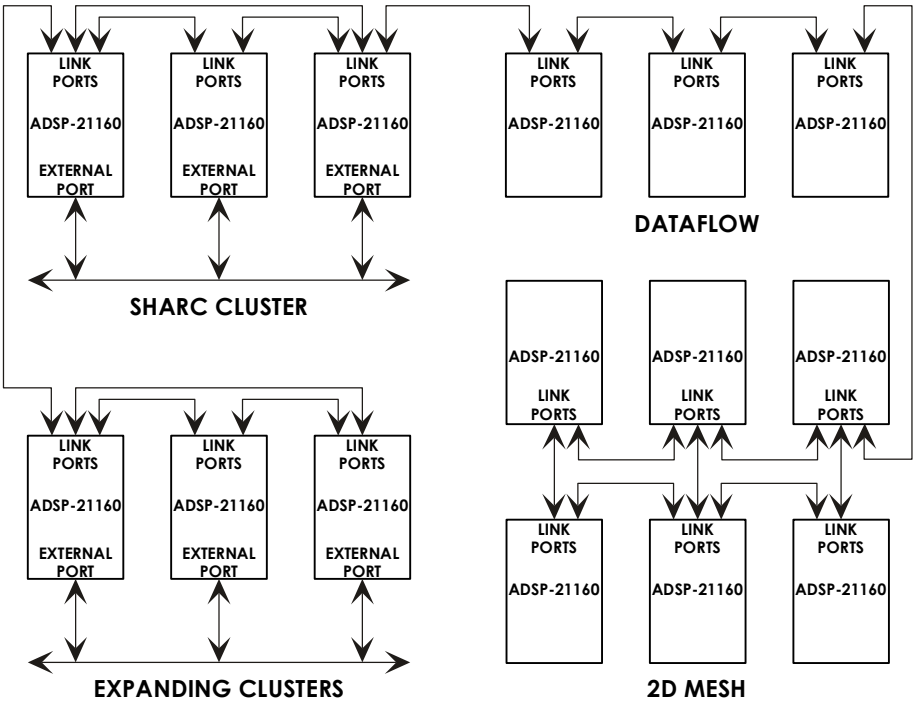


Figure 8-7. Link Port Communication Examples

9 SERIAL PORTS

This chapter describes ADSP-21160 DSP's serial ports.

Overview

The DSP has two independent, synchronous serial ports, SPORT0 and SPORT1, that provide an I/O interface to a wide variety of peripheral devices. Each serial port has its own set of control registers and data buffers. With a range of clock and frame synchronization options, the SPORTs allow a variety of serial communication protocols and provide a glueless hardware interface to many industry-standard data converters and CODECs.because channels 0 through

The serial ports can operate at 1/2 the full clock rate of the processor, providing each with a maximum data rate of n/2 Mbit/s, where n equals the processor clock frequency. Independent transmit and receive functions provide greater flexibility for serial communications. Serial port data can be automatically transferred to and from on-chip memory using DMA block transfers. Each of the serial ports offers a TDM (time division multiplexed) multichannel mode.

Serial port clocks and frame syncs can be internally generated by the DSP or received from an external source. The serial ports can operate with little-endian or big-endian transmission formats, with word lengths selectable from 3 to 32 bits. They offer selectable synchronization and transmit modes and optional μ-law or A-law companding in hardware.

The serial ports offer the following features and capabilities:

- Provides independent transmit and receive functions
- Transfers data words up to 32 bits in length, either MSB-first or LSB-first
- Double-buffers data—both receive and transmit functions have a data buffer register and a shift register—the double-buffering provides additional time to service the SPORT
- Compands (compression/decompression) A-law and μ-law hardware companding on transmitted and received words.
- Internally generates serial clock and frame sync signals—in a wide range of frequencies—or accepts clock and frame synch input from an external source
- Performs interrupt-driven, single-word transfers to and from on-chip memory controlled by the DSP core
- Executes DMA transfers to and from on-chip memory—each SPORT can automatically receive and transmit an entire block of data
- Permits chaining of DMA operations for multiple data blocks
- Has a multichannel mode for TDM interfaces—each SPORT can receive and transmit data selectively from channels of a time-division-multiplexed serial bitstream—this mode can be useful for T1 interfaces

Table 9-1 shows the pins of each serial port:

Table 9-1. Serial Port Pins

SPORT0 Pins	SPORT1 Pins	Description
DT0	DT1	Transmit Data
TCLK0	TCLK1	Transmit Clock
TFS0	TFS1	Transmit Frame Sync
DR0	DR1	Receive Data
RCLK0	RCLK1	Receive Clock
RFS0	RFS1	Receive Frame Sync

A serial port receives serial data on its DR input and transmits serial data on its DT output. It can receive and transmit simultaneously for full duplex operation.

Serial communications are synchronized to a clock signal—every data bit must be accompanied by a clock pulse. Each serial port can generate or receive its own transmit clock signal (TCLK) and receive clock signal (RCLK). Internally-generated serial clock frequencies are configured in the TDIVx and RDIVx registers.

In addition to the serial clock signal, data may be signalled by a frame synchronization signal. The framing signal can occur either at the beginning of an individual word or at the beginning of a block of words. The configuration of frame synch signals depends upon the type of serial device connected to the DSP. Each serial port can generate or receive its own transmit frame sync signal (TFS) and receive frame sync signal (RFS). Internally-generated frame sync frequencies are configured in the TDIVx and RDIVx registers.

Figure 9-1 shows a block diagram of a serial port. Data to be transmitted is written to the TX buffer. The data is (optionally) compressed in hardware, then automatically transferred to the transmit shift register. The

data in the shift register is then shifted out on the SPORT's DT pin, synchronous to the TCLK transmit clock. If framing signals are used, the TFS signal indicates the start of the serial word transmission. The DT pin is always driven (for example, three-stated) if the serial port is enabled (SPEN=1 in the STCTLx control register), unless it is in multichannel mode and an inactive time slot occurs. For more information, see "Multichannel Operation" on page -28.

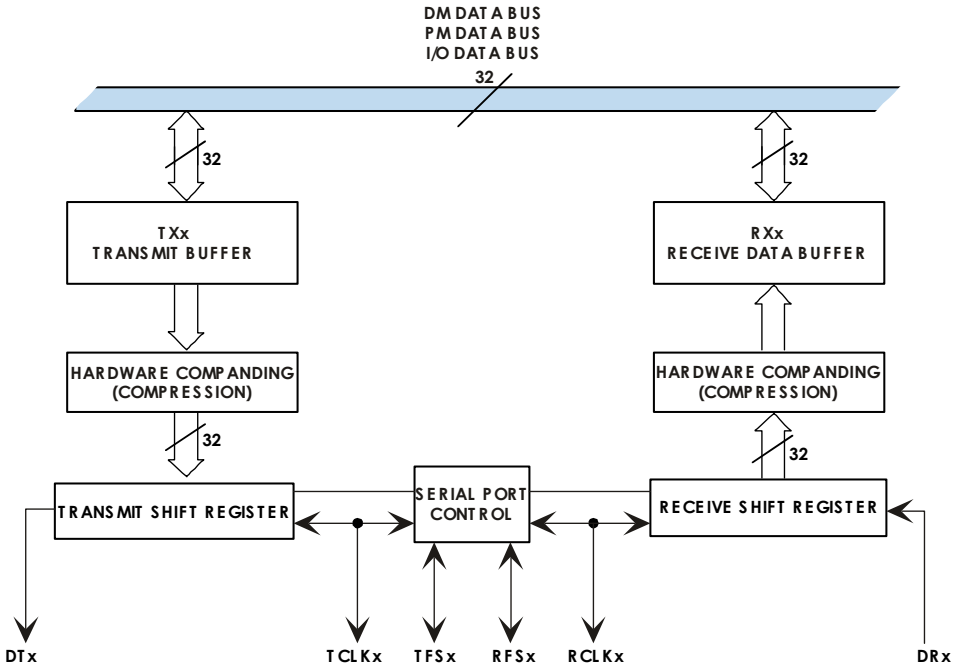


Figure 9-1. Serial Port Block Diagram

The receive portion of the SPORT shifts in data from the DR pin, synchronous to the RCLK receive clock. If framing signals are used, the RFS signal indicates the beginning of the serial word being received. When an entire

word is shifted in, the data is (optionally) expanded, then automatically transferred to the RX buffer.



The DSP SPORTs are not UARTs and cannot be used to communicate with an RS-232 device or any other asynchronous communications protocol. One way to implement RS-232-compatible communications with the DSP is to use two of the FLAG pins as asynchronous data receive and transmit signals. For an example of how to do this, see Chapter 11 "*Software UART*" in the *Digital Signal Processing Applications Using The ADSP-2100 Family*, Volume 2.

SPORT Interrupts

Each serial port has a transmit DMA interrupt and a receive DMA interrupt. When serial port DMA is not enabled, interrupts occur based on the SPORT transmit or receive FIFO status. If on the transmit side the FIFO is empty or on the receive side the FIFO is full, interrupts are generated. The priority of the serial port interrupts is shown in Table 9-2.

Table 9-2. SPORT Interrupts

Interrupt Name	Interrupt
SPR0I	SPORT0 Receive DMA Channel Highest Priority
SPR1I	SPORT1 Receive DMA Channel
SPT0I	SPORT0 Transmit DMA Channel
SPT1I	SPORT1 Transmit DMA Channel Lowest Priority
The interrupt names are defined in the def21160.h include file supplied with the ADSP-21xxx DSP Development Software.	



SPORT Interrupts occur on the second system clock (CLKIN) after the last bit of the serial word is latched in or driven out.

SPORT Reset

There are two ways to reset the serial ports: a hardware reset using the RESET pin of the processor, and a software reset accomplished by clearing the serial port's enable bit (SPEN) in the STCTLx and SRCTLx control registers. Each method has a different effect on the serial port.

A hardware reset disables the serial ports by clearing the STCTLx and SRCTLx control registers (including the SPEN enable bits) and the TDIVx and RDIVx frame sync divisor registers. Any ongoing operations are aborted.

A software reset of the SPEN enable bit(s) disables the serial port(s) and aborts any ongoing operations. Status bits are also cleared. The serial ports are ready to start transmitting or receiving data two CLKIN cycles after they are enabled (in the STCTLx or SRCTLx control register). No serial clocks are be lost from this point on.



The only difference between the soft (setting the SRST bit in SYSCON register and hard reset (RESET pin) is that the external bus arbitration does not get affected by a soft reset. That is, there is no bus synchronization at soft reset. The PLL also does not get reset at soft reset.

Setting Serial Port Modes

The registers used to control and configure the serial ports are part of the IOP register set. Each SPORT has its own set of the control registers and data buffers, as shown in Table 9-3.

Table 9-3. SPORT Registers

Register Name*	Function
STCTLx	SPORT Transmit Control Register
TXx	Transmit Data Buffer
TDIVx	Transmit Clock and Frame Sync Divisors
MTCSx	Multichannel Transmit Select
MTCCSx	Multichannel Transmit Compand Select
SRCTLx	SPORT Receive Control Register
RXx	Receive Data Buffer
RDIVx	Receive Clock and Frame Sync Divisors
MRCSx	Multichannel Receive Select
MRCCSx	Multichannel Receive Compand Select
SPATHx	SPORT Path Length (for mesh multiprocessing)
KEYWDx	SPORT Receive Comparison
KEYMASKx	SPORT Receive Comparison Mask
An asterisk (*) indicates x = 0, 1.	

These control registers are describe in detail in the following sections:

- “SPORT Serial Transmit Control Registers (STCTLx)” on page A-71
- “SPORT Serial Receive Control Registers (SRCTLx)” on page A-73

- "SPORT Transmit Buffer Registers (TXx)" on page A-73
- "SPORT Receive Buffer Registers (RXx)" on page A-76
- "SPORT Transmit Divisor Registers (TDIVx)" on page A-77
- "SPORT Transmit Count Registers (TCNTx)" on page A-78
- "SPORT Receive Divisor Registers (RDIVx)" on page A-78
- "SPORT Receive Count Registers (RCNTx)" on page A-79
- "SPORT Transmit Select Registers (MTCSx)" on page A-79
- "SPORT Receive Select Registers (MRCSx)" on page A-80
- "SPORT Transmit Compand Registers (MTCCSx)" on page A-80
- "SPORT Receive Compand Register (MRCCSx)" on page A-80
- "SPORT Receive Comparison and Mask Registers (KEYWDx and KEYMASKx)" on page A-81
- "SPORT Serial Path Length Registers (SPATHx)" on page A-81

These sections show the memory-mapped address and reset initialization value of each SPORT control register. All of the registers are 32 bits wide.

The SPORT control registers are programmed by writing to the appropriate address in memory. The symbolic names of the registers and individual control bits can be used in DSP programs—the `#define` definitions for these symbols are contained in the file `def21160.h` which is provided in the `INCLUDE` directory of the ADSP-21xxx DSP Development Software. The `def21160.h` file is shown in the Control/Status Registers appendix of this manual. All control and status bits in the SPORT registers are active high unless otherwise noted.

Because the SPORT registers are memory-mapped they cannot be written with data coming directly from memory. They must instead be written

from (or read into) DSP core registers, usually one of the general-purpose universal registers of the register file (R15-R0). For instance, the SPORT control registers can also be written or read by external devices (for example, another DSP or a host processor) to set up a serial port DMA operation.

Transmit and Receive Control Registers (STCTL, SRCTL)

The main control registers for each serial port are the transmit control register, STCTLx, and the receive control register, SRCTLx. These registers are defined in Table A-27 on page A-71 and Table A-28 on page A-74. When changing operating modes, a serial port control register should be cleared (for example, written with all zeros) before the new mode is written to the register.

The Transmit Underflow Status bit (TUVF) is set whenever the TFS signal occurs (from either external or internal source) while the TX buffer is empty. The internally generated TFS may be suppressed whenever TX is empty by clearing the DITFS control bit (DITFS=0).

When DITFS=0, the default, the transmit frame sync signal (TFS) is dependent upon new data being present in the TX buffer—the TFS signal is only generated for new data. Setting DITFS to 1 selects data-independent frame syncs. This causes the TFS signal to be generated whether or not new data is present, transmitting the contents of the TX buffer regardless. Serial port DMA typically keeps the TX buffer full, and when the DMA operation is complete the last word in TX is continuously transmitted.

The TXS status bits indicate whether the TX buffer is full (11), empty (00), or partially full (10). To test for space in TX, test for TXS0 (bit 30) equal to zero. To test for the presence of any data in TX, test for TXS1 (bit 31) equal to one.

The SRCTLx and STCTLx registers control the serial ports operating modes for the I/O processor. Table A-28 on page A-74 lists all the bits in SRCTLx and Table A-27 on page A-71 lists all the bits in STCTLx.

The following bits control serial port modes. Some other bits in the SRCTLx and STCTLx registers are used to setup DMA and I/O processor related serial port features. For information on these features, see "Setting I/O Processor—SPort Modes" on page 6-49.

- **Internal Transmit Clock Select.** SRCTLx Bit 10 (ICLK) This bit selects the internal receive clock (if set, =1) or external receive clock (if cleared, =0).
- **Clock Rising Edge Select.** SRCTLx Bit 12 (CKRE) This bit select whether the serial port uses the rising edge (if set, =1) or falling edge (if cleared, =0) of the clock signal for sampling data and the frame sync.
- **Receive Frame Sync Required Select.** SRCTLx Bit 13 (RFSR) This bit selects whether the serial port requires (if set, =1) or does not require (if cleared, =0) a receive frame synch.
- **Internal Receive Frame Sync Select.** SRCTLx Bit 14 (IRFS) This bit selects whether the serial port uses an internal RFS (if set, =1) or uses an external RFS (if cleared, =0).
- **Data Independent Receive Frame Sync Select.** SRCTLx Bit 15 (DIRFS) This bit selects whether the serial port uses a data-independent RFS (synch at selected interval, if set, =1) or uses a data-dependent RFS (synch when data in RX, if cleared, =0).
- **Active Low Receive Frame Synch Select.** SRCTLx Bit 16 (LRFS) This bit selects an active low RFS (if set, =1) or active high RFS (if cleared, =0).

- **Late Receive Frame Sync Select.** SRCTLx Bit 17 (LAFS) This bit selects a late RFS (RFS during first bit, if set, =1) or an early RFS (RFS before first bit, if cleared, =0). This bit must be cleared for multichannel operation.
- **Serial Port Loopback Enable.** SRCTLx Bit 22 (SPL) This bit enables (if set, =1) or disables (if cleared, =0) serial port loopback mode. This bit must be cleared for multichannel operation.
- **Multichannel Enable.** SRCTLx Bit 23 (MCE) This bit enables (if set, =1) or disables (if cleared, =0) multichannel serial port mode.
- **Number of Multi Channels (–1) Select.** SRCTLx Bits 28-24 (NCHN) These bits select the number of channels (–1) for a multichannel serial port. The number of channels can be from 1 (NCHN=0) to 32 (NCHN=31).
- **Transmit Frame Sync Required Select.** STCTLx Bit 13 (TFSR) This bit selects whether the serial port requires (if set, =1) or does not require (if cleared, =0) a transfer frame synch.
- **Internal Transmit Frame Sync Select.** STCTLx Bit 14 (ITFS) This bit selects whether the serial port uses an internal TFS (if set, =1) or uses an external TFS (if cleared, =0).
- **Data Independent Transmit Frame Sync Select.** STCTLx 15 (DITFS) This bit selects whether the serial port uses a data-independent TFS (synch at selected interval, if set, =1) or uses a data-dependent TFS (synch when data in TX, if cleared, =0).
- **Active Low Transmit Frame Synch Select.** STCTLx Bit 16 (LTFS) This bit selects an active low TFS (if set, =1) or active high TFS (if cleared, =0).

- **Late Transmit Frame Sync Select.** STCTLx Bit 17 (LAFS) This bit selects a late TFS (TFS during first bit, if set, =1) or an early TFS (TFS before first bit, if cleared, =0).
- **Multichannel Transmit Frame Sync Delay Select.** STCTLx Bits 23-20 (MFD) These bits select the delay in serial clock cycles between the TFS and the first data bit. When MFD=0, the TFS and first data bit are concurrent. The maximum value is MFD=16.
- **Current Channel Selected (read-only).** STCTLx Bits 28-24 (CHNL) These bits indicate which channel the DSP has selected for the serial port's transmission in multichannel mode.

Register Writes and Effect Latency

SPORT register writes are internally completed at the end of the same CLKIN cycle in which they occur. The register is read back the newly written value on the very next cycle. When a read of one of the STCTLx or SRCTLx control registers is immediately followed by a write to that register, the write may take two cycles to complete.

After a write to a SPORT register, control and mode bit changes generally take effect in the second CLKIN cycle after the write is completed. The serial ports are ready to start transmitting or receiving two CLKIN cycles after they are enabled (in the STCTLx or SRCTLx control register). No serial clocks are lost from this point on.

Transmit and Receive Data Buffers (TX, RX)

TX0 and TX1 are the transmit data buffers for SPORT0 and SPORT1. They are 32-bit buffers which must be loaded with the data to be transmitted; the data is loaded either by the DMA controller or by the program running on the DSP core. RX0 and RX1 are the receive data buffers for SPORT0 and SPORT1. They are 32-bit buffers which are automatically loaded from the receive shifter when a complete word has been received.

Word lengths of less than 32 bits are right-justified in the receive and transmit buffers.

The TX buffers act like a two-location FIFO because they have a data register plus an output shift register as shown in Figure 9-1 on page 9-4. Two 32-bit words may be stored in TX at any one time. When the TX buffer is loaded and any previous word has been transmitted, the buffer contents are automatically loaded into the output shifter. An interrupt is generated when the output shifter has been loaded, signifying that the TX buffer is ready to accept the next word (for example, the TX buffer is "not full"). This interrupt does not occur if serial port DMA is enabled or if the corresponding mask bit in the IMASK register is set.

The transmit underflow status bit (TUVF) is set in the transmit control register when a transmit frame synch occurs and no new data has been loaded into TX. The TUVF status bit is "sticky" and is only cleared by disabling the serial port.

The RX buffers act like a three-location FIFO because they have two data registers plus an input shift register. Two complete 32-bit words can be stored in RX while a third word is being shifted in. The third word overwrites the second if the first word has not been read out (by the DSP core or the DMA controller). When this happens, the receive overflow status bit (ROVF) is set in the receive control register. Almost three complete words can be received without the RX buffer being read before overflow occurs. The overflow status is generated on the last bit of third word. The ROVF status bit is "sticky" and is only cleared by disabling the serial port.

An interrupt is generated when the RX buffer has been loaded with a received word (for example, the RX buffer is "not empty"). This interrupt is masked out if serial port DMA is enabled or if the corresponding bit in the IMASK register is set.

If your DSP program causes the core processor to attempt a read from an empty RX buffer or a write to a full TX buffer, the access is delayed until the buffer is accessed by the external I/O device. (This delay is called a core

processor hang.) If it is not known whether the core processor can access the RX or TX buffer without a hang, the buffer's full or empty status should be read first (in STCTLx or SRCTLx) to determine if the access can be made.



To support debugging buffer transfers, the DSP has a Buffer Hang Disable (BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD discussion on page -18.

The status bits in STCTLx and SRCTLx are updated during reads and writes from the core processor even when the serial port is disabled. The serial port should be disabled when writing to the RX buffer or reading from the TX buffer.

Clock and Frame Sync Frequencies (TDIV, RDIV)

The TDIVx and RDIVx registers contain divisor values which determine the frequencies for internally generated clocks and frame syncs. These registers are defined in "SPORT Transmit Divisor Registers (TDIVx)" on page A-77 and "SPORT Receive Divisor Registers (RDIVx)" on page A-78.

$$f_{RCLK} = \frac{f_{CCLK}}{2(RCLKDIV + 1)}$$

$$f_{TCLK} = \frac{f_{CCLK}}{2(TCLKDIV + 1)}$$

The maximum serial clock frequency is equal to 1/2 the DSP's internal clock (CCLK) frequency, which occurs when xCLKDIV is set to zero. Use the

following equation to determine the value of xCLKDIV to use, given the CCLK frequency and desired serial clock frequency:

$$RCLKDIV = \frac{f_{CCLK}}{2(f_{RCLK})} - 1$$

$$TCLKDIV = \frac{f_{CCLK}}{2(f_{TCLK})} - 1$$

The DSP's internal clock (CCLK) is the CLKIN frequency multiplied by a clock ratio (CLK_CFG3-0). For more information, see the clock ratio discussion on page 11-8.

TFSDIV and RFSDIV specify how many transmit or receive clock cycles are counted before generating a TFS or RFS pulse (when the frame synch is internally generated). In this way a frame sync can be used to initiate periodic transfers. The counting of serial clock cycles applies to either internally or externally generated serial clocks.

The formula for the number of cycles between frame synch pulses is:

```
# of serial clocks between frame syncs = xFSDIV + 1
```

Use the following equations to determine the value of xFSDIV to use, given the serial clock frequency and desired frame sync frequency:

$$TFSDIV = \frac{f_{TCLK}}{f_{TFS}} - 1$$

$$RFSDIV = \frac{f_{RCLK}}{f_{RFS}} - 1$$

The frame sync is continuously active if xFSDIV=0. The value of xFSDIV should not be less than the serial word length minus one (the value of the SLEN field in the transmit or receive control register), as this may cause an external device to abort the current operation or cause other unpredictable results. If the serial port is not being used, the xFSDIV divisor can be used as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. The serial port must be enabled for this mode of operation to work.



Caution should be exercised when operating with externally generated transmit clocks near the frequency of 1/2 the DSP's internal clock. There is a delay between when the clock arrives at the TCLKx pin and when data is output—this delay may limit the receiver's

speed of operation. Refer to the data sheet for exact timing specifications. For reliable operation, it is recommended that full-speed serial clocks only be used when receiving with an externally generated clock and externally generated frame sync (ICLK=0, IRFS=0).

Externally-generated late transmit frame syncs also experience a delay from when they arrive to when data is output—this can also limit the maximum serial clock speed. Refer to the data sheet for exact timing specifications.

The serial ports handle word lengths of 3 to 32 bits, but transmitting or receiving words smaller than 7 bits at 1/2 the full clock rate of the DSP may cause incorrect operation when DMA chaining is enabled. Chaining disables the DSP's internal I/O bus for several cycles while the new TCB parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

Data Word Formats

The format of the data words transmitted over the serial ports is configured by the DTYPE, SENDN, SLEN, and PACK bits of the STCTLx and SRCTLx control registers.

Word Length

The serial ports handle word lengths of 3 to 32 bits. The word length is configured in the 5-bit SLEN field in the STCTLx and SRCTLx control registers. The value of SLEN is equal to the word length minus one:

```
SLEN = Serial Word Length - 1
```

The SLEN value should not be set to zero or one. Words smaller than 32 bits are right-justified in the RX and TX buffers, residing in the least significant bit positions.

Transmitting or receiving words smaller than 7 bits at 1/2 the full clock rate of the DSP may cause incorrect operation when DMA chaining is

enabled. Chaining disables the DSP's internal I/O bus for several cycles while the new TCB parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

Endian Format

Endian format determines whether the serial word is transmitted MSB-first or LSB-first. Endian format is selected by the SENDN bit in the STCTLx and SRCTLx control registers. When SENDN=0, serial words are transmitted (or received) MSB-first. When SENDN=1, serial words are transmitted (or received) LSB-first.

Data Packing and Unpacking

Received data words of 16 bits or less may be packed into 32-bit words, and 32-bit words being transmitted may be unpacked into 16-bit words. Word packing and unpacking is selected by the PACK bit in the SRCTLx and STCTLx control registers.

When PACK=1 in the receive control register (SRCTLx), two successive words received are packed into a single 32-bit word.

When PACK=1 in the transmit control register (STCTLx), each 32-bit word is unpacked and transmitted as two 16-bit words.

The first 16-bit (or smaller) word is right-justified in bits 15-0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31-16. This applies for both receive (packing) and transmit (unpacking) operations. Companding may be used when word packing or unpacking is being used.

When serial port data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.



When 16-bit received data is packed into 32-bit words and stored in normal word space in DSP internal memory, the 16-bit words can be read or written with short word space addresses.

Data Type

The DTYPE field of the STCTLx and SRCTLx control registers specifies one of four data formats (for non-multichannel operation):

Table 9-4. DTYPE and Data Formatting (non-multichannel)

DTYPE	Data Formatting
00	Right-justify, zero-fill unused MSBs
01	Right-justify, sign-extend into unused MSBs
10	Compand using μ-law
11	Compand using A-law

These formats are applied to serial data words loaded into the RX and TX buffers. TX data words are not actually zero-filled or sign-extended, because only the significant bits are transmitted.

For multichannel operation, the companding selection and MSB-fill selection is independent:

Table 9-5. DTYPE and Data Formatting (multichannel)

DTYPE	Data Formatting
x0	Right-justify, zero-fill unused MSBs
x1	Right-justify, sign-extend into unused MSBs
0x	Compand using μ-law
1x	Compand using A-law

Linear transfers occur if the channel is active but companding is not selected for that channel. Companded transfers occur if the channel is active and companding is selected for that channel. The multichannel compand select registers, MTCCSx and MRCCSx, are used to specify which

transmit and receive channels are companded. For more information, see "Channel Selection Registers" on page 9-32.

Transmit sign extension is selected by bit 0 of DTYPE in the STCTLx register and is common to all transmit channels. Receive sign extension is selected by bit 0 of DTYPE in the SRCTLx register and is common to all receive channels. If bit 0 of DTYPE is set, sign extension occurs on selected channels that do not have companding selected. If this bit is not set, the word contains 0s in the MSBs.

Companding

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The DSP serial ports support the two most widely used companding algorithms, A-law and μ-law, performed according to the CCITT G.711 specification. The type of companding can be selected independently for each SPORT. Companding is selected by the DTYPE field of the STCTLx and SRCTLx control registers.

When companding is enabled, the data in the RX0 or RX1 buffer is the right-justified, sign-extended expanded value of the eight LSBs received. A write to TX0 or TX1 causes the 32-bit value to be compressed to eight LSBs (sign-extended to the width of the transmit word) before it is transmitted. If the 32-bit value is greater than the 13-bit A-law or 14-bit μ-law maximum, it is automatically compressed to the maximum value.

Because the values in the TX and RX buffers are actually companded in-place, the companding hardware can be used without transmitting (or receiving) any data, for example during testing or debugging. This operation requires a single cycle of overhead, as described below. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compand data in-place, without transmitting, use the following sequence of operations:

1. Enable companding in the DTYPE field of the STCTLx transmit control register.
2. Write a 32-bit data word to TX. (The companding is calculated in this cycle.)
3. Wait one cycle. A NOP instruction can be used to do this; if a NOP is not inserted, the DSP core is held off for one cycle anyway. This allows the serial port companding hardware to reload TX with the companded value.
4. Read the 8-bit companded value from TX.

To expand data in-place, the same sequence of operations is used but with RX rather than TX. When expanding data in this way, be sure that the serial word length (SLEN) is set appropriately in the SRCTLx control register.

With companding enabled, interfacing the DSP serial port to a codec requires little additional programming effort. If companding is not selected, there are two formats available for received data words of fewer than 32 bits: one that fills unused MSBs with zeros, and another that sign-extends the MSB into the unused bits. For more information, see "Data Type" on page 9-19.

Clock Signal Options

Each serial port has a transmit clock signal (TCLKx) and a receive clock signal (RCLKx). The clock signals are configured by the ICLK and CKRE bits of the STCTLx and SRCTLx control registers. Serial clock frequency is configured in the TDIVx and RDIVx registers.



The receive clock pin may be tied to the transmit clock if a single clock is desired for both input and output.

Both transmit and receive clocks can be independently generated internally or input from an external source. The ICLK bit of the STCTLx and SRCTLx control registers determines the clock source.

When ICLK=1, the clock signal is generated internally by the DSP and the TCLKx or RCLKx pins are outputs. The clock frequency is determined by the value of the serial clock divisor (TCLKDIV or RCLKDIV) in the TDIVx or RDIVx registers.

When ICLK=0, the clock signal is accepted as an input on the TCLKx or RCLKx pins, and the serial clock divisors in the TDIVx/RDIVx registers are ignored. The externally generated serial clock need not be synchronous with the DSP system clock.

Frame Sync Options

Framing signals indicate the beginning of each serial word transfer. The framing signals for each serial port are TFS (transmit frame synchronization) and RFS (receive frame synchronization). A variety of framing options are available; these options are configured in the serial port control registers. The TFS and RFS signals of a serial port are independent and are separately configured in the control registers.

Framed Versus Unframed

The use of frame sync signals is optional in serial port communications. The TFSR (transmit frame sync required) and RFSR (receive frame sync required) control bits determine whether frame sync signals are required. These bits are located in the STCTLx and SRCTLx control registers.

When TFSR=1 or RFSR=1, a frame sync signal is required for every data word. To allow continuous transmitting from the DSP, each new data word must be loaded into the TX buffer before the previous word is shifted out and transmitted. For more information, see “Data-Independent Transmit Frame Sync” on page 9-27.

When TFSR=0 or RFSR=0, the corresponding frame sync signal is not required. A single frame sync is needed to initiate communications but is ignored after the first bit is transferred. Data words are then transferred continuously, unframed.

When DMA is enabled in this mode, with frame syncs not required, DMA requests may be held off by chaining or may not be serviced frequently enough to guarantee continuous unframed data flow.

Figure 9-2 illustrates framed serial transfers, which have the following characteristics:

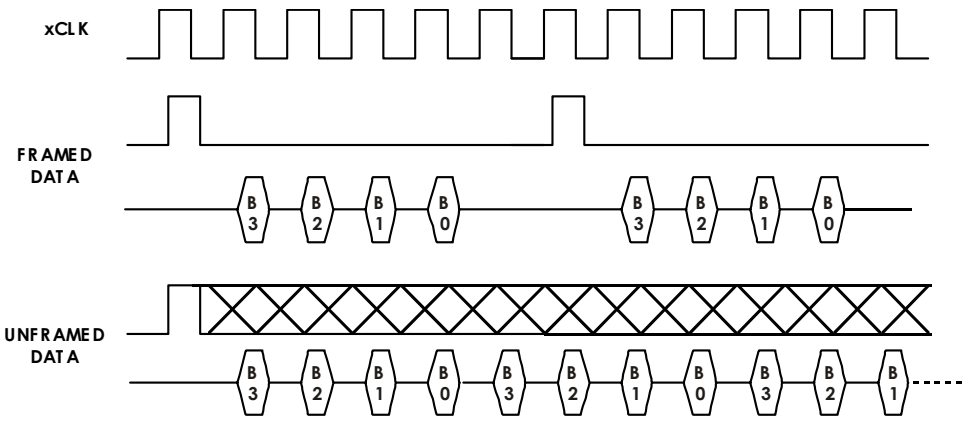


Figure 9-2. Framed Versus Unframed Data

- TFSR and RFSR bits in STCTLx, SRCTLx control registers determine framed or unframed mode.
- Framed mode requires a framing signal for every word. Unframed mode ignores framing signal after first word.
- Unframed mode is appropriate for continuous reception.
- Active-low or active-high frame syncs selected with LTFS and LRFS bits of STCTLx, SRCTLx control registers.

Internal Versus External Frame Syncs

Both transmit and receive frame syncs can be independently generated internally or input from an external source. The ITFS and IRFS bits of the STCTLx and SRCTLx control registers determine the frame sync source.

When ITFS=1 or IRFS=1, the corresponding frame sync signal is generated internally by the DSP and the TFSx pin or RFSx pin is an output. The frequency of the frame sync signal is determined by the value of the frame sync divisor (TFSDIV or RFSDIV) in the TDIVx or RDIVx registers.

When ITFS=0 or IRFS=0, the corresponding frame sync signal is accepted as an input on the TFSx pin or RFSx pins, and the frame sync divisors in the TDIVx/RDIVx registers are ignored.

All of the various frame sync options are available whether the signal is generated internally or externally.

Active Low Versus Active High Frame Syncs

Frame sync signals may be either active high or active low (for example, inverted). The LTFS and LRFS bits of the STCTLx and SRCTLx control registers determine the frame syncs' logic level:

- When LTFS=0 or LRFS=0, the corresponding frame sync signal is active high.
- When LTFS=1 or LRFS=1, the corresponding frame sync signal is active low.

Active high frame syncs are the default. The LTFS and LRFS bits are initialized to 0 after a processor reset.

Sampling Edge For Data and Frame Syncs

Data and frame syncs can be sampled on either the rising or falling edges of the serial port clock signals. The CKRE bit of the STCTLx and SRCTLx control registers selects the sampling edge.

For transmit data and frame syncs, setting CKRE=1 in STCTLx selects the rising edge of TCLKx. CKRE=0 selects the falling edge. Note that data and frame sync signals change state on the clock edge that is not selected.

For receive data and frame syncs, setting CKRE=1 in SRCTLx selects the rising edge of RCLKx. CKRE=0 selects the falling edge.

The transmit and receive functions of two serial ports connected together, for example, should always select the same value for CKRE so that any internally generated signals are driven on one edge and any received signals are sampled on the opposite edge.

Early Versus Late Frame Syncs

Frame sync signals can occur during the first bit of each data word ("late") or during the serial clock cycle immediately preceding the first bit ("early"). The LAFS bit of the STCTLx and SRCTLx control registers configures this option.

When LAFS=0, early frame syncs are configured; this is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted, and the frame sync is not checked again until the entire word has been transmitted (or received). (In multichannel operation, this is the case when frame delay is 1.)

If data transmission is continuous in early framing mode (for example, the last bit of each word is immediately followed by the first bit of the next word), then the frame sync signal occurs during the last bit of each word. Internally generated frame syncs are asserted for one clock cycle in early framing mode.

When LAFS=1, late frame syncs are configured; this is the alternate mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the same serial clock cycle that the frame sync is asserted. (In multichannel operation, this is the case when frame delay is zero.) Receive data bits are latched by serial clock edges, but the frame sync signal is only checked during the first bit of each word. Internally generated frame syncs remain asserted for the entire length of the data word in late framing mode. Externally generated frame syncs are only checked during the first bit.

Figure 9-3 illustrates the two modes of frame signal timing:

- LAFS bits of STCTLx, SRCTLx control registers. LAFS=0 for early frame syncs, LAFS=1 for late frame syncs.
- Early framing: frame sync precedes data by one cycle. Late framing: frame sync checked on first bit only.
- Data transmitted MSB-first (SENDN=0) or LSB-first (SENDN=1).
- Frame sync and clock generated internally or externally.

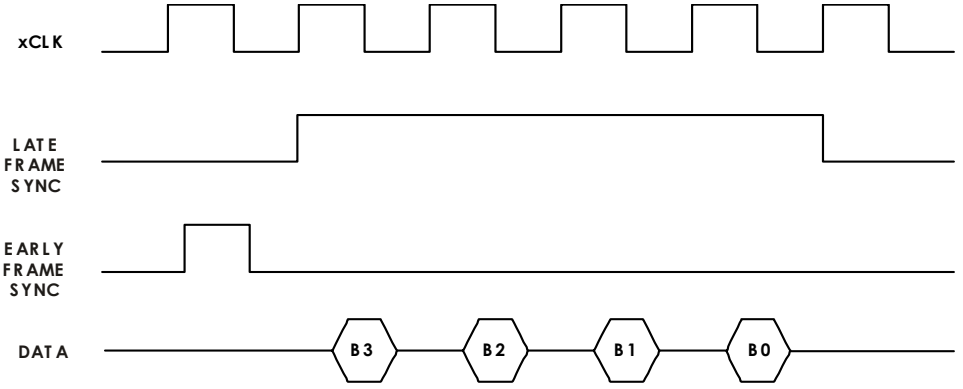


Figure 9-3. Normal Versus Alternate Framing

Data-Independent Transmit Frame Sync

Normally the internally generated transmit frame sync signal (TFS) is output only when the TX buffer has data ready to transmit. The DITFS mode (data-independent transmit frame sync) allows the continuous generation of the TFS signal, with or without new data. The DITFS bit of the STCTLx control register configures this option.

When DITFS=0, the internally generated TFS is only output when a new data word has been loaded into the TX buffer. Once data is loaded into TX, it is not transmitted until the next TFS is generated. This mode of operation allows data to be transmitted only at specific times.

When DITFS=1, the internally generated TFS is output at its programmed interval regardless of whether new data is available in the TX buffer. Whatever data is present in TX is retransmitted with each assertion of TFS. The TUVF transmit underflow status bit (in the STCTLx control register) is set when this occurs (for example, when old data is retransmitted). The TUVF status bit is also set if the TX buffer does not have new data when an externally generated TFS occurs. Note that in this mode of operation, the first internally generated TFS is delayed until data has been loaded into the TX buffer.

If the internally generated TFS is used, a single write to the TX data register is required to start the transfer.

SPORT Loopback

When the SPL bit (SPORT loopback) is set in the SRCTLx receive control register, the serial port is configured in an internal loopback connection. The loopback configuration allows the serial ports to be tested internally.

When loopback is configured, the DRx, RCLKx, and RFSx signals of the receive section of the SPORT are internally connected to the DTx, TCLKx, and TFSx signals of the transmit section.

The DTx, TCLKx, and TFSx signals are active and are available at their respective pins, while the DRx, RCLKx, and RFSx pins are ignored by the DSP.



Only transmit clock and transmit frame sync options may be used in loopback mode—programs must ensure that the serial port is set up correctly in the STCTLx and SRCTLx control registers. Multichannel mode is not allowed.

Multichannel Operation

The DSP serial ports offer a multichannel mode of operation which allows the SPORT to communicate in a time-division-multiplexed (TDM) serial system. In multichannel communications, each data word of the serial bit stream occupies a separate channel— each word belongs to the next consecutive channel so that, for example, a 24-word block of data contains one word for each of 24 channels.

The serial port can automatically select words for particular channels while ignoring the others. Up to 32 channels are available for transmitting or receiving—each SPORT can receive and transmit data selectively from any of the 32 channels. In other words, the SPORT can do any of the following on each channel:

- transmit data
- receive data
- transmit and receive data, or
- do nothing

Data companding and DMA transfers can also be used in multichannel mode.

The DT pin is always driven (for example, not three-stated) if the serial port is enabled (SPEN=1 in the STCTLx control register), unless it is in multichannel mode and an inactive time slot occurs.

Note that (in multichannel mode) the TCLKx pin is always an input and must be connected to its corresponding RCLKx pin.

Figure 9-4 shows example timing for a multichannel transfer, which have the following characteristics:

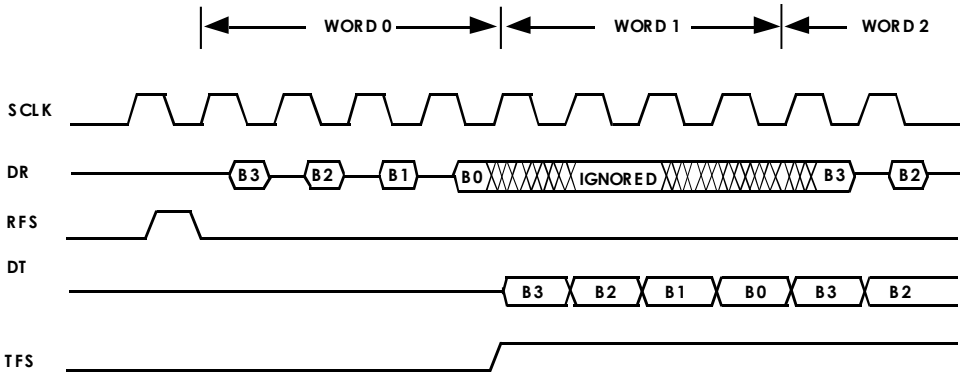


Figure 9-4. Multichannel Operation

- Uses TDM method where serial data is sent or received on different channels sharing the same serial bus.
- The number of channels is selected with the NCH bits of SRCTLx:

  ```
  NCH=(# of channels) - 1
  ```

- Can independently select transmit and receive channels.
- RFS signal start of frame.
- TFS is used as “Transmit Data Valid” for external logic; active only during transmit channels.
- Example: Receive on channels 0 and 2. Transmit on channels 1 and 2.

Frame Syncs in Multichannel Mode

All receiving and transmitting devices in a multichannel system must have the same timing reference. The RFS signal is used for this reference, indicating the start of a block (or frame) of multichannel data words.

When multichannel mode is enabled on a SPORT, both the transmitter and receiver use RFS as a frame sync. This is true whether RFS is generated internally or externally. The RFS signal is used to synchronize the channels and restart each multichannel sequence. RFS assertion occurs the beginning of the channel 0 data word.

TFS is used as a transmit data valid signal which is active during transmission of an enabled word. Because the serial port's DTx pin is three-stated when the time slot is not active, the TFS signal specifies whether or not DTx is being driven by the DSP. The DSP drives TFS in multichannel mode whether or not ITFS is cleared.

After the TX transmit buffer is loaded, transmission begins and the TFS signal is generated. When serial port DMA is being used, this may happen several cycles after the multichannel transmission is enabled. If a deterministic start time is required, the TX buffer should be preloaded.



TFS is normally left unconnected in multichannel mode, and the RFS pins of the serial port(s) are usually connected together.

Multichannel Control Bits in STCTL, SRCTL

The STCTLx and SRCTLx control registers contain several bits used to enable and configure multichannel operations. Multichannel mode is enabled by setting the MCE bit in the SRCTLx control register:

- When MCE=1, multichannel operation is enabled.
- When MCE=0, all multichannel operations are disabled.

Multichannel operation is activated three cycles after MCE is set. Internally generated frame sync signals activate four cycles after MCE is set.

Setting the MCE bit enables multichannel operation for both receive and transmit sides of the SPORT. A transmitting SPORT must be in multichannel mode if the receiving SPORT is in multichannel mode.

The SLEN bits determine the serial bit length of the transmit and receive data words. The SLEN bit settings in the STCTLx register (bits 4-8) should match the SLEN bit settings in the SRCTLx register.

The number of channels used in multichannel operation is selected by the 5-bit NCH field in the SRCTLx control register. NCH should be set to the actual number of channels minus one:

```
NCH = Number of Channels - 1
```

The 5-bit CHNL field in the STCTLx control register indicates which channel is currently selected during multichannel operation. This field is a read-only status indicator. CHNL(4:0) increments modulo NCH(4:0) as each channel is serviced.

The 4-bit MFD field in the STCTLx control register specifies a delay between the frame sync pulse and the first data bit in multichannel mode. The value of MFD is the number of serial clock cycles of the delay. Multichannel frame delay allows the processor to work with different types of T1 interface devices.

A value of zero for MFD causes the frame sync to be concurrent with the first data bit. The maximum value allowed for MFD is 15. A new frame sync may occur before data from the last frame has been received, because blocks of data occur back to back.

A multichannel frame delay of at least one should be used when the DSP is generating frame syncs for the multichannel system and the serial clock of the system is equal to CLKIN (the processor clock). If MFD is not set to at least one, the master DSP in a multiprocessing system does not recognize

the first frame sync after multichannel operation is enabled. All succeeding frame syncs are recognized normally.

Channel Selection Registers

Specific channels can be individually enabled or disabled to select which words are received and transmitted during multichannel communications. Data words from the enabled channels are received or transmitted, while disabled channel words are ignored. Up to 32 channels are available for transmitting and up to 32 channels for receiving.

The multichannel selection registers are used to enable and disable individual channels. The registers for each serial port are as shown in Table 9-6.

Table 9-6. Multichannel Selection Registers

Register Name	Function
MTCSx	Multichannel Transmit Select—specifies the active transmit channels
MRCSx	Multichannel Receive Select—specifies the active receive channels
MTCCSx	Multichannel Transmit Compand Select—specifies which active transmit channels are companded
MRCCSx	Multichannel Receive Compand Select—specifies which active receive channels are companded

Each register has 32 bits, corresponding the 32 channels. Setting a bit enables that channel so that the serial port selects its word from the multiple-word block of data (for either receive or transmit). For example, setting bit 0 selects word 0, setting bit 12 selects word 12, and so on.

Setting a particular bit to 1 in the `MTCSx` register causes the serial port to transmit the word in that channel's position of the data stream. Clearing the bit to 0 in the `MTCSx` register causes the serial port's `DT` (data transmit) pin to three-state during the time slot of that channel.

Setting a particular bit to 1 in the MRCSx register causes the serial port to receive the word in that channel's position of the data stream; the received word is loaded into the RX buffer. Clearing the bit to 0 in the MRCSx register causes the serial port to ignore the data.

Companding may be selected on a per-channel basis. The MTCCSx and MRCCSx registers are used to specify companding for any active channels. Setting a bit to 1 in these registers causes the data to be companded. A-law or μ-law companding is selected with the DTYPE bit 1 in the STCTLx and SRCTLx control registers.

SPORT Receive Comparison Registers

On the DSP, two sets of registers aid multiprocessor communications when using multichannel mode (MCE=1) through the serial ports. These 32-bit registers are the Receive Comparison (KEYWDx) registers and the Receive Comparison Mask (KEYMASKx) registers.

Table 9-7 shows the MCE setting as well as the bits in the SRCTL register that control the operation of Receive Comparison.

Table 9-7. Receive Comparison Selection

IMODE (Bit 15)	IMAT (Bit 20)	Operation
0	x	Receive comparison disabled
1	0	Accept receive data if the KEYWD comparison is false
1	1	Accept receive data if the KEYWD comparison is true

The KEYWD0 or KEYWD1 register stores the pattern to be matched with the incoming data. The corresponding KEYMASK0 or KEYMASK1 register specifies which of the bits in the received data should be compared. Setting KEYMASKx bit (=1) masks the corresponding bit in KEYWDx register, disabling its comparison.

The processor receiving the data compares it with the data in the KEYWDx register. Depending on the comparison results, the received data is accepted or ignored. If accepted, the receiver requests—based on the setting to the SRCTL register—a DMA transfer to internal memory or generates an interrupt.

When receive comparison is enabled, companding is disabled on the transmitter and receiver. The MTCCSx register, which selects multichannel companding when receive comparison is disabled, determines whether the DSP performs a KEYWD comparison for the enabled received channels. If the MTCCSx bit for a particular channel is '0,' the processor does not perform a comparison and always accepts the receive data on that channel. If the MTCCSx bit for a particular channel is '1,' the processor performs the comparison and accepts (or rejects) the receive data, depending on the result of the comparison and IMAT setting in the SRCTLx register.

The receive comparison feature lets the DSP's SPORTs generate a DMA request or an interrupt when the received data matches a specified condition on a specified channel in multichannel mode. Without this feature, the SPORT would interrupt the processor every time data was received and the processor would be required to check if the data was meant for it or not. It is possible that most of the time the data being sent is not meant for the processor. With the receive comparison feature, the SPORT on a particular processor can be programmed to interrupt only on messages meant for that processor.

As a receive comparison example, consider four DSPs (A, B, C, and D) which use SPORT0 (in multichannel mode) for interprocessor communication. Channels 0, 1, 2, and 3 are used respectively by A, B, C, and D to transmit control information between the processors. Channels 4 through 10, 11 through 17, 18 through 24, and 25 through 31 are used respectively by A, B, C, and D to transmit data.

Because channels 0 through 3 are used to send control information between the processors, the comparisons for incoming data is enabled only for these channels. Initially, channels 4 through 31 may have receive dis-

abled. For this example, consider communication between processors A and B only. The keyword for comparison is programmable; in this example, processor B can check for the keyword `START TRANSMIT TO B`. Processor B can check for this keyword as follows.

1. Set the `KEYWD` register to `START TRANSMIT TO B`.
2. Clear bits 31:16 of the `KEYMASK` register to 0 and set the other bits to 1.

   This step enables comparison only for bits 31:16. So, assume that the code for `START TRANSMIT TO B` only uses bits 31:16 and bits 15:0 indicate the source of the transmission and the data channels.
3. Set bits 15 and 20 of the `SRCTL` register to 1.

   This step enables the SPORT to generate an interrupt or DMA request only if the incoming data matches the `KEYWD`.
4. Set bits 0 through 3 of the Transmit Compand Channel Selector register to 1 and clear the remaining bits to 0.

   This step enables comparison only on channels 0 through 3.

Until it receives the `START TRANSMIT TO B` keyword, processor B ignores all transmissions that it receives. When processor A wants to send data to B, it sends this keyword on channel 0. When receive comparison on processor B recognizes the `START TRANSMIT TO B` keyword, the SPORT interrupts processor B. Then, processor B analyzes the remaining 16-bits, determining that the source is processor A and the data is on channels 4 through 10.

Because processor A is using channels 4 through 10 to transmit data, processor B enables receive channels 4 through 10 and sends a “READY TO RECEIVE DATA” message to processor A, using channel 1. After processor A receives this message, it sends the data on channels 4 through 10.

If the transfer protocol uses a fixed number of bytes in each message, processor B could send back a checksum message to processor A after receiving A's message, confirming that the data transferred accurately.

Moving Data Between SPORTS and Memory

Transmit and receive data can be transferred between the DSP serial ports and on-chip memory in one of two ways, with single-word transfers or with DMA block transfers. Both methods are interrupt-driven, using the same internally generated interrupts.

When serial port DMA is not enabled in the STCTLx or SRCTLx control registers, the SPORT generates an interrupt every time it has received a data word or has started to transmit a data word. SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the interrupt is generated. The DSP's on-chip DMA controller handles the DMA transfer, allowing the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

DMA Block Transfers

The DSP's on-chip DMA controller allows automatic DMA transfers between internal memory and the two serial ports. There are four DMA channels for serial port operations—each SPORT has one channel for receiving data and one for transmitting data. The serial port DMA channels are numbered as follows:

- DMA Channel 0 – SPORT0 Receive
- DMA Channel 1 – SPORT1 Receive

- DMA Channel 2 – SPORT0 Transmit
- DMA Channel 3 – SPORT1 Transmit

The SPORT DMA channels are assigned higher priority than all other DMA channels (for example, link ports and the external port) because of their relatively low service rate and their inability to hold off incoming data. Having higher priority causes the SPORT DMA transfers to be performed first when multiple DMA requests occur in the same cycle.

Although the DMA transfers are always performed with 32-bit words, the serial ports can handle word sizes from 3 to 32 bits. If the serial words are 16 bits or smaller, they can be packed into 32-bit words for each DMA transfer; this is configured by the PACK bit of the STCTLx and SRCTLx control registers. When serial port data packing is enabled (PACK=1), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

The following sections present an overview of serial port DMA operations; some additional details are covered in the DMA chapter of this manual.

- For information on SPORT DMA Channel Setup, see "Setting up Serial Port DMA" on page 6-91.
- For information on SPORT DMA Parameter Registers, see "Setting I/O Processor—SPort Modes" on page 6-49.
- For information on SPORT DMA Chaining, see "Chaining DMA Processes" on page 6-70.

Single-Word Transfers

Individual data words may also be transmitted and received by the serial ports, with interrupts occurring as each 32-bit word is transmitted or received. When a serial port is enabled and DMA is disabled (in the STCTLx or SRCTLx control registers), the SPORT DMA interrupts are generated in this way—whenever a complete 32-bit word has been received in

the RX buffer, or whenever the TX buffer is not full. Single-word interrupts can be used to implement interrupt-driven I/O on the serial ports.

Whenever the DSP core's program reads a word from a serial port's RX buffer or writes a word to its TX buffer, the buffer's full/empty status should first be checked in order to avoid hanging the DSP core. (This can also happen to an external device, for example a host processor, when it is reading or writing a serial port buffer.) The full/empty status can be read in the RXS bits of the SRCTLx register or the TXS bits of the STCTLx register. Reading from an empty RX buffer or writing to a full TX buffer causes the DSP (or external device) to hang, waiting for the status to change.



To support debugging buffer transfers, the DSP has a Buffer Hang Disable (BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see the BHD discussion on page -18.

Multiple interrupts can occur if both SPORTs transmit or receive data in the same cycle. Any interrupt can be masked out in the IMASK register; if the interrupt is later enabled in IMASK, the corresponding interrupt latch bit in IRPTL must be cleared in case the interrupt has occurred in the meantime.

When serial port data packing is enabled (PACK=1 in the STCTLx or SRCTLx control registers), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

SPORT Pin/Line Terminations

The DSP has very fast drivers on all output pins including the serial ports. If connections on the data, clock, or frame sync lines are longer than six inches, you should consider using a series termination for strip lines on point-to-point connections. This may be necessary even when using low-speed serial clocks, because of the edge rates.

10 JTAG TEST EMULATION PORT

A boundary scan allows a system designer to test interconnections on a printed circuit board with minimal test-specific hardware. The scan is made possible by the ability to control and monitor each input and output pin on each chip through a set of serially scannable latches. Each input and output is connected to a latch, and the latches are connected as a long shift register so that data can be read from or written to them through a serial test access port (TAP).

The ADSP-21160 DSP contains a test access port compatible with the industry-standard IEEE 1149.1 (JTAG) specification. Only the IEEE 1149.1 features specific to the ADSP-21160 DSPs are described here. For more information, see the IEEE 1149.1 specification and other the documents listed in "References" on page 10-38.

The boundary scan allows a variety of functions to be performed on each input and output signal of the ADSP-21160 DSP. Each input has a latch that monitors the value of the incoming signal and can also drive data into the chip in place of the incoming value. Similarly, each output has a latch that monitors the outgoing signal and can also drive the output in place of the outgoing value. For bidirectional pins, the combination of input and output functions is available.

Every latch associated with a pin is part of a single serial shift register path. Each latch is a master/slave type latch with the controlling clock provided externally. This clock (`TCK`) is asynchronous to the ADSP-21160 system clock (`CLKIN`).

The ADSP-21160 emulation features let you halt the processor at a pre-defined point. You then examine the state of the processor, execute arbitrary code, restore the original state, and continue execution.



The ADSP-21160 emulation features are a superset of the ADSP-21060 emulation features. All emulation features supported by previous SHARC DSPs are supported on the ADSP-21160 DSP, except the ICSA output signal and function. The set of features on which EZ-ICE® designs rely are supported in an identical fashion on ADSP-21160 DSP. The ADSP-21160 DSP can be used with the ADSP-2106x SHARC EZ-ICE hardware.

There are several changes/extensions to the base functionality of the ADSP-21060 emulation capability, which require changes in the EZ-ICE software for ADSP-21160 DSP support. These extensions include:

- The emulation breakpoint address start/end registers have moved from UREG space to IOP register space. This change did not effect the TSTEMU block directly, only the address decodes to gain access to it.
- EMU64PX has been added to the IR decode space. This shift register provides access to the full 64-bit wide PX register of ADSP-21160 DSP.
- A memory test shift register has been added to the IR decode space. This feature is for Analog Devices internal use ONLY.
- Addition of the MTST (Memory TeST) bit in the EMUCTL register. This feature is for Analog Devices internal use ONLY. EMUCTL is 40 bits wide on the ADSP-21160 DSP.

Several on chip facilities are directly accessed through the JTAG interface. These facilities are listed in Table 10-2 on page 10-4. Other emulation facilities are only indirectly accessible. To indirectly access the facilities that do not appear in Table 10-2, scan the instruction which moves data of interest to/from the PX register, scan the PX data (if the instruction is a

PX read), let the core execute the instruction, then scan the PX register out (if the instruction was a PX write).

The breakpoint start/end registers are mapped into the IOP register space of the ADSP-21160 DSP. For specific addresses, see "Register and Bit #Defines File (def21160.h)" on page A-82. The EMUN, EMUCLK, and EMUCLK2 registers occupy the same UREG address space as on the ADSP-2106 DSPs.

These facilities are read only by the ADSP-21160 processor core in normal operation.

JTAG Test Access Port

The emulator uses JTAG boundary scan logic for ADSP-21160 communications and control. This JTAG logic consists of a state machine, a five pin Test Access Port (TAP), and shift registers. The state machine and pins conform to the IEEE 1149.1 specification. The TAP pins appear in Table 10-1.

Table 10-1. JTAG Test Access Port (TAP) Pins

Pin	Function
TCK	(input) Test Clock: pin used to clock the TAP state machine.[1]
TMS	(input) Test Mode Select: pin used to control the TAP state machine sequence.[2]
TDI	(input) Test Data In: serial shift data input pin.
TDO	(output) Test Data Out: serial shift data output pin.
$\overline{\text{TRST}}$	(input) Test Logic Reset: resets the TAP state machine

1 Asynchronous with CLKIN
2 Synchronous to CLKIN

A BSDL file for the ADSP-21160 DSP is available on Analog Devices' website. Set your browser to:

```
http://www.analog.com/dsp
```

Refer to the IEEE 1149.1 JTAG specification for detailed information on the JTAG interface. The many sections of this appendix assume a working knowledge of the JTAG specification.

Instruction Register

The instruction register allows an instruction to be shifted into the processor. This instruction selects the test to be performed and/or the test data register to be accessed. The instruction register is 5 bits long with no parity bit. A value of 10000 binary is loaded (LSB nearest `TDO`) into the instruction register whenever the TAP reset state is entered.

Table 10-2 lists the binary code for each instruction. Bit 0 is nearest `TDO` and bit 4 is nearest `TDI`. No data registers are placed into test modes by any of the public instructions. The instructions affect the ADSP-21160 DSP as defined in the 1149.1 specification. The optional instructions `RUNBIST`, `IDCODE` and `USERCODE` are not supported by the ADSP-21160 DSP.

Table 10-2. JTAG Instruction Register Codes

43210	Register	Instruction	Comment	Type
11111	Bypass	BYPASS		Public
00000	Boundary	EXTEST		Public
10000	Boundary	SAMPLE		Public
01000	EMUPMD	EMULATION	48-bit scan length	Private
11000	Boundary	INTEST		Public
00100	EMUCTL	EMULATION		Private
10100	EMUPX	EMULATION	48-bit shift register	Private

Table 10-2. JTAG Instruction Register Codes (Cont'd)

43210	Register	Instruction	Comment	Type
10110	EMU64PX	EMULATION	64-bit shift register	Private
01100	EMUSTAT	EMULATION		Private
11100	BRKSTAT	EMULATION		Private
00010	EMUPC	EMULATION		Private
10101	MEMTST	TEST	Memory test	Private
All others	Reserved	Reserved		Private

The entry under "Register" is the serial scan path, either Boundary or Bypass in this case, enabled by the instruction. Figure 10-1 on page 10-6 shows these register paths. The 1-bit Bypass register is fully defined in the 1149.1 specification. For more information on the Boundary register, see "Boundary Register" on page 10-17.

No special values need be written into any register prior to selection of any instruction. As Table 10-2 on page 10-4 shows, certain instructions are reserved for emulator use. For more information, see Table 10-7 on page 10-19.

EMUPMD Shift Register

The EMUPMD serial shift register is located in the system unit. EMUPMD is 48 bits wide and is accessed by the emulator through TAP. When the TAP enters the UPDATE state and EMUPMD is selected, a 48-bit slave register is updated from EMUPMD. EMUPMD's purpose is to force the ADSP-21160 DSP to execute emulator supplied instructions. The register accomplishes this by driving the instruction bus while in emulation space.

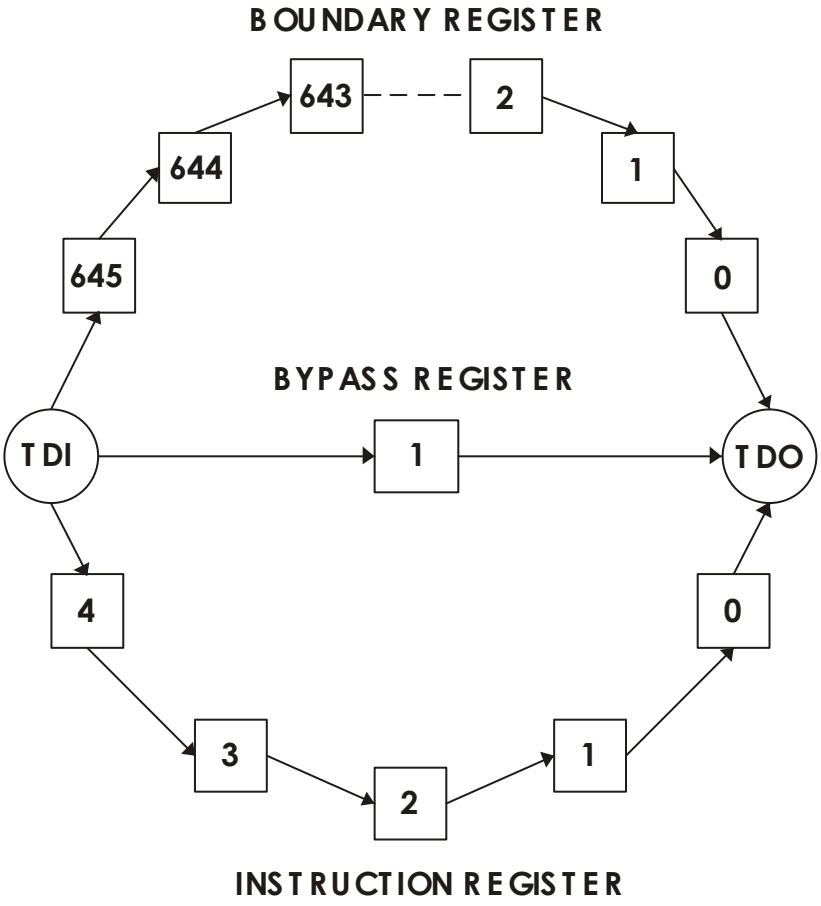


Figure 10-1. Serial Scan Paths

EMUPX Shift Register

The EMUPX serial shift register is located in the system unit. EMUPX is a 48-bits wide and is accessed by the emulator through the TAP. When the TAP goes into the UPDATE state and EMUPX is selected, the most significant 48-bits of PX is updated from EMUPX. When the TAP goes into the CAPTURE state and EMUPX is selected, EMUPX is updated with the most sig-

nificant 48-bits of PX. The EMUPX register is used to transfer data between the emulator and the target system.

EMUPX is provided for backwards compatibility with the SHARC ICE hardware. PX is a 64-bit wide register. To provide compatibility, only the most significant 48 bits of PX are mapped to EMUPX. 48-bit instructions, and 40-bit extended precision data, are always aligned to the most significant bit. When transferring 32-bit data to/from PX register, PX2 must be specified as the source/destination to ensure that the 32 bits is aligned to the most significant bit.

EMU64PX Shift Register

The EMU64PX serial shift register is located in the system unit. EMU64PX is 64-bits wide and is accessed by the emulator through the TAP. When the TAP goes into the UPDATE state and EMU64PX is selected, PX is updated from EMU64PX. When the TAP goes into the CAPTURE state and EMU64PX is selected, EMU64PX is updated from PX. The EMU64PX register transfers data between the emulator and the target system. The most significant 48 bits of EMU64PX are redundantly available in EMUPX.

EMUPC Shift Register

The EMUPC serial shift register is located in the system unit. EMUPC is 24-bits wide and is accessed by the emulator through the TAP. The EMUPC register captures addresses from the PC register. This data can be used to statistically profile the user's code. Addresses cannot be forced into the PC register from the EMUPC register.

EMUCTL Shift Register

The EMUCTL serial shift register is located in the system unit. EMUCTL is 40-bits wide and is accessed by the emulator through the TAP. EMUCTL

controls all of the ADSP-21160 emulation functionality. Table 10-3 lists EMUCTL's bits and describes their functionality.

Table 10-3. EMUCTL (Emulation Control) Register Definition

Bit #	Name	Function
0	EMUENA	Emulator Function Enable. The EMUENA bit enables ADSP-21160 emulation functions. (0=ignore breakpoints and emulator interrupts, 1=respond to breakpoints and emulator interrupts)
1	EIRQENA	Emulator Interrupt Enable. The EIRQENA bit enables the emulation logic to recognize external emulator interrupts. (0=disable, 1=enable)
2	BKSTOP	Enable Autostop on Breakpoint. The BKSTOP bit enables the ADPS-21160 DSP to generate an external emulator interrupt when any breakpoint event occurs. (0=disable, 1=enable)
3	SS	Enable Single Step Mode. The SS bit enables single-step operation. (0=disable, 1=enable)
4	SYSRST	Software Reset of the ADSP-21160 DSP. The SYSRST bit resets the ADSP-21160 DSP in the same manner as the external $\overline{\text{RESET}}$ pin. The SYSRST bit must be cleared by the emulator.(0=normal operation, 1=reset)
5	ENBRKOUT	Enable the $\overline{\text{BRKOUT}}$ pin. The ENBRKOUT bit enables the $\overline{\text{BRKOUT}}$ pin operation. (0=$\overline{\text{BRKOUT}}$ pin at high-impedance state, 1=$\overline{\text{BRKOUT}}$ pin enabled)
6	IOSTOP	Stop IOP DMAs in EMU space. The IOSTOP bit disables all DMA requests when the DSP is in emulation space. Data that is currently in the EP, LINK, or SPORT DMA buffers is held there unless the internal DMA request was already granted. IOSTOP causes incoming data to be held off and outgoing data to cease. Because SPORT receive data cannot be held off, it is lost and the overrun bit is set. The direct write buffer (internal memory write) and the EP pad buffer are allowed to flush any remaining data to internal memory. (0=IO continues, 1=IO Stops)

Table 10-3. EMUCTL (Emulation Control) Register Definition (Cont'd)

Bit #	Name	Function
7	EPSTOP	Stop I/O Processor EP operation in emulation space. The EPSTOP bit disables all EP requests when the DSP is in emulation space. After an emulation interrupt is acknowledged, EPSTOP deasserts ACK (deasserts REDY if host access) to prevent further data from being accepted if the EP is accessed. The emulator may clear this bit—allowing I/O to continue and the bus to clear—so that the emulator may use the EP (through BR and bus lock). Note that the EP bus clears only if accesses are direct writes or IOP register writes, because all other IOP functions are halted. The EP bus does not clear if accesses to any of the DMA buffers are extended due to a buffer full or empty condition. (0=EP IO continues, 1=EP IO Stops)
8	NEGPA1	Negate program memory data address breakpoint. The NEG bits enable breakpoint events if the address is greater than the end register value OR less than the start register value. This function is useful to detect index range violations in user code. (0=disable breakpoint, 1=enable breakpoint)
9	NEGDA1	Negate data memory address breakpoint #1. For more information, see NEGPA1 bit description on page 10-9.
10	NEGDA2	Negate data memory address breakpoint #2. For more information, see NEGPA1 bit description on page 10-9.
11	NEGIA1	Negate instruction address breakpoint #1. For more information, see NEGPA1 bit description on page 10-9.
12	NEGIA2	Negate instruction address breakpoint #2. For more information, see NEGPA1 bit description on page 10-9.
13	NEGIA3	Negate instruction address breakpoint #3. For more information, see NEGPA1 bit description on page 10-9.
14	NEGIA4	Negate instruction address breakpoint #4. For more information, see NEGPA1 bit description on page 10-9.
15	NEGIO1	Negate I/O address breakpoint. For more information, see NEGPA1 bit description on page 10-9.
16	NEGEP1	Negate EP address breakpoint. For more information, see NEGPA1 bit description on page 10-9.

Table 10-3. EMUCTL (Emulation Control) Register Definition (Cont'd)

Bit #	Name	Function
17	ENBPA	Enable program memory data address breakpoints. The ENB bits enable each breakpoint group. Note that when the ANDBKP bit is set, breakpoint types not involved in the generation of the effective breakpoint must be disabled. (0=disable breakpoints, 1=enable breakpoints)
18	ENBDA	Enable data memory address breakpoints. For more information, see ENBPA bit description on page 10-10.
19	ENBIA	Enable instruction address breakpoints. For more information, see ENBPA bit description on page 10-10.
20	ENBIO	Enable I/O address breakpoint. For more information, see ENBPA bit description on page 10-10.
21	ENBEP	Enable external port address breakpoint. For more information, see ENBPA bit description on page 10-10.
22-23	PA1MODE	PA1 breakpoint triggering mode. The breakpoint triggering mode bits trigger on the following conditions: **ModeTriggering condition** 00Breakpoint is disabled 01WRITE accesses only 10READ accesses only 11any access
24-25	DA1MODE	DA1 breakpoint triggering mode. For more information, see PA1MODES bit description on page 10-10.
26-27	DA2MODE	DA2 breakpoint triggering mode. For more information, see PA1MODES bit description on page 10-10.
28-29	IO1MODE	IO1 breakpoint triggering mode. For more information, see PA1MODES bit description on page 10-10.
30-31	EP1MODE	EP1 breakpoint triggering mode. For more information, see PA1MODES bit description on page 10-10.
32	ANDBKP	AND composite breakpoints. The ANDBKP bit enables ANDing of each breakpoint type to generate an effective breakpoint from the composite breakpoint signals. (0=OR breakpoint types, 1=AND breakpoint types)
33	RESERVED	The ICSA function and DMDSEL bit used by that function not supported on ADSP-21160 DSP.

Table 10-3. EMUCTL (Emulation Control) Register Definition (Cont'd)

Bit #	Name	Function
34	NOBOOT	No power-up boot on reset. The NOBOOT bit forces the ADSP-21160 DSP into the No boot mode. In this mode, the processor does not boot load, but begins fetching instructions from 0x0080 0004 in external memory. (0=disable, 1=force No boot mode)
35	TMODE	Test mode enable. The TMODE bit is for Analog Devices' usage only. Do NOT set this bit. (0=normal operation)
36	BHO	Buffer Hang Override bit. The BHO control bit overrides the BHD bit in SYSCON, disabling BHD's control over core access of data buffer behavior. Note that the default (reset) state of BHD is now set for ADSP-21160 DSP, a change from ADSP-2106x DSPs. (0=normal BHD operation, 1=override BHD operation)
37	MTST	Memory Test Enable Bit. The MTST bit enables scanning of data for to the latches used for memory test. (0=normal operation, 1=enable memory test)
38, 39	Reserved	Reserved

EMUSTAT Shift Register

The EMUSTAT serial shift register is located in the system unit. EMUSTAT is 8-bits wide and is accessed by the emulator through the TAP. This register is updated by the ADSP-21160 DSP when the TAP is in the CAPTURE state.

The emulator reads EMUSTAT to determine the state of the ADSP-21160 DSP. None of the bits in this register can be written by the emulator. All bits are active high. Table 10-4 lists the EMUSTAT register's bits.

Table 10-4. EMUSTAT (Emulation Status) Register Definition

Bit	Name	Function (If bit=1...)
0	EMUSPACE	Indicates that the next instruction is to be fetched from the emulator.
1	EMUREADY	Indicates that the ADSP-21160 has finished executing the previous emulator instruction.
2	INIDLE	Indicates that the ADSP-21160 was in IDLE prior to the latest emulator interrupt.
3	COMHALT	Indicates a core access to a SPORT or a LINK is hung because of an external device.
4	EPHALT	Indicates a core access to a DMA buffer is hung because of the external port.
5-7	Reserved	

BRKSTAT Shift Register

The BRKSTAT serial shift register is located in the system unit. BRKSTAT is a 16 bits wide and is accessed by the emulator through the TAP. This register monitors the status of the emulation breakpoints and is updated on every clock cycle. None of the bits of this register can be written by the emulator. Table 10-5 lists the BRKSTAT register's bits.

A high bit indicates a breakpoint hit. When a breakpoint hit occurs, the register ceases updating. Stopping allows the emulator to see which breakpoint was triggered. When the ADSP-21160 DSP leaves emulation space

the BRKSTAT register is cleared and resumes updating. All status bits are synchronized to TCLK before being scanned out.

Table 10-5. BRKSTAT (Breakpoint Status) Register Definition

Bit #	Name	Function (If bit=1...)
0	STATPA	Program Memory Data breakpoint hit
1	STATDA0	Data Memory breakpoint hit
2	STATDA1	Data Memory breakpoint hit
3	STATIA0	Instruction Address breakpoint hit
4	STATIA1	Instruction Address breakpoint hit
5	STATIA2	Instruction Address breakpoint hit
6	STATIA2	Instruction Address breakpoint hit
7	STATIO	I/O Address breakpoint hit
8	STATEP	EP Address breakpoint hit
9-15	Reserved	

MEMTST Shift Register

The MEMTST serial shift register is for Analog Devices' usage only.



Do not attempt to use this register—incorrect usage of this feature can result in permanent damage to the ADSP-21160 DSP being tested.

PSx, DMx, IOx, and EPx (Breakpoint) Registers

The PSx, DMx, IOx, and EPx (Breakpoint) registers are located in the I/O Processor register set. The emulation breakpoint registers are not user accessible and can be written only when the ADSP-21160 DSP is in emulation space or test mode. The breakpoint registers vary in size according to the address type: instruction (24-bit address), data (32-bit address), or I/O data (19-bit address). Table 10-6 shows the sizes.

The ADSP-21160 DSP contains nine sets of emulation breakpoint registers. Each set consists of a start and end register which describe an address range, with the start register setting the lower end of the address range. Each breakpoint set monitors a particular address bus. When a valid address is in the address range, than a breakpoint signal is generated. The address range includes the start and end addresses.

The nine breakpoint sets are grouped into five types: instruction (IA), DM data (DA), PM data (PA), IO data (IO), and EP data (EP). The individual breakpoint signals in each type are ORed together to create five composite breakpoint signals.

These composite signals can be optionally ANDed or ORed together to create the effective breakpoint event signal used to generate an emulator interrupt. The ANDBKP bit in the EMUCTL register selects the function used.

Each breakpoint type has an enable bit in the EMUCTL register. When set, these bits add the specified breakpoint type into the generation of the effective breakpoint signal. If cleared, the specified breakpoint type is not used in the generation of the effective breakpoint signal. This allows the user to trigger the effective breakpoint from a subset of the breakpoint types.

To provide further flexibility, each individual breakpoint can be programmed to trigger if the address is in range AND one of these conditions is met: READ access, WRITE access, ANY access, or NO access. The control bits for this feature are also located in EMUCTL. For more information, see PA1MODES bit description on page 10-10.

The address ranges of the emulation breakpoint registers are negated by setting the appropriate renege negation bits in the EMUCTL register. For more information, see NEGPA1 bit description on page 10-9. Each breakpoint can be disabled by setting the start address larger than the end address.

Four of the breakpoints monitor the instruction address. Two monitor the data memory address. One monitors the program memory data address, one monitors the I/O address bus and one monitors the EP address bus.

The instruction address breakpoints monitor the address of the instruction being executed, not the address of the instruction being fetched. If the current execution is aborted, the breakpoint signal does not occur even if the address is in range. Data address breakpoints (DA and PA only) are also ignored during aborted instructions.

The nine breakpoint sets appear in Table 10-6.

Table 10-6. PSx, DMx, IOx, and EPx (Breakpoint) Registers

Register	Function	Group[1]
PSA1S	Instruction Address Start #1	IA
PSA1E	Instruction Address End #1	IA
PSA2S	Instruction Address Start #2	IA
PSA2E	Instruction Address End #2	IA
PSA3S	Instruction Address Start #3	IA
PSA3E	Instruction Address End #3	IA
PSA4S	Instruction Address Start #4	IA
PSA4E	Instruction Address End #4	IA
DMA1S	Data Address Start #1	DA
DMA1E	Data Address End #1	DA
DMA2S	Data Address Start #2	DA
DMA2E	Data Address End #2	DA
PMDAS	Program Data Address Start	PA
PMDAE	Program Data Address End	PA
IOAS	I/O Address Start	IO
IOAE	I/O Address End	IO

Table 10-6. PSx, DMx, IOx, and EPx (Breakpoint) Registers (Cont'd)

Register	Function	Group[1]
EPAS	External Port Address Start	EP
EPAE	External Port Address End	EP

1 Group IA=24-bit addresses, Groups DA, PA, and EP=32-bit addresses, Group IO=19-bit addresses.

EMUN Register

The EMUN (Nth event counter) register is located in the I/O Processor register set. The EMUN register is not user accessible and can be written only when the ADSP-21160 DSP is in emulation space. EMUN is read-only from normal-space and can be written only when the ADSP-21160 DSP is in emulation space. The Nth event counter allows an emulation breakpoint to occur on the Nth occurrence of the breakpoint event. This is accomplished by writing the desired Nth value to the EMUN register in UREG space. This register can be read from normal space, but it can be written only in emulation space. The counter decrements on each occurrence of the breakpoint event, asserting the interrupt when the counter is equal to zero and the hardware breakpoint event occurs.

EMUCLK and EMUCLK2 Registers

The EMUCLK (clock counter) and EMUCLK2 (clock counter scaling) registers are located in the universal (UREG) register set. EMUCLK and EMUCLK2 are not user accessible and can be written only when the ADSP-21160 DSP is in emulation space. These registers are read-only from normal-space and can be written only when the ADSP-21160 DSP is in emulation space.

The Emulation Clock Counter consists of a 32-bit count register (EMUCLK) and a 32-bit scaling register (EMUCLK2). The EMUCLK counts clock cycles while the user has control of the ADSP-21160 DSP and stops counting when the emulator gains control. These registers let you gauge the amount

of time spent executing a particular section of code. The EMUCLK2 register extends the time EMUCLK can count by incrementing each time the EMUCLK value rolls over to zero. The combined emulation clock counter can count accurately for thousands of hours.

EMUIDLE Instruction

The EMUIDLE instruction places the ADSP-21160 DSP in the IDLE state and triggers an emulator interrupt. This operation lets you use the EMUIDLE instruction to be used as a software breakpoint. When EMUIDLE is executed, the emulation clock counter immediately halts.

In-Circuit Signal Analyzer (ICSA) Function

This function is NOT supported in ADSP-21160 DSP.

Boundary Register

The Boundary register is 655 bits long. Table 10-7 defines the latch type and function of each position in the scan path. The positions are numbered with 0 being the first bit output (closest to TDO) and 654 being the last (closest to TDI).

Notes on Boundary registers:

- Scan position 0 (LODAT0) is the end is closest to TDO (scan in first)
- Scan position 654 (SPARE); this end is closest to TDI (scan in last)

- Output Enables:
  - 1 = Drive the associated signals during the EXTEST and INTEST instructions
  - 0 = Three-state the associated signals during the EXTEST and INTEST instructions

  CLKIN can be sampled but not controlled (read-only). CLKIN continues to clock the ADSP-21160DSP no matter which instruction is enabled.

Table 10-7. JTAG Boundary Register

Scan #	Signal Name	Latch Type
0	L0DAT(0)	Output
1	L0DAT(0)	Output enable
2	L0DAT(0)	Input
3	L0DAT(1)	Output
4	L0DAT(1)	Output enable
5	L0DAT(1)	Input
6	L0DAT(2)	Output
7	L0DAT(2)	Output enable
8	L0DAT(2)	Input
9	L0DAT(3)	Output
10	L0DAT(3)	Output enable
11	L0DAT(3)	Input
12	L0ACK	Output
13	L0ACK	Output enable
14	L0ACK	Input
15	L0CLK	Output
16	L0CLK	Output enable
17	L0CLK	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
18	L0DAT(4)	Output
19	L0DAT(4)	Output enable
20	L0DAT(4)	Input
21	L0DAT(5)	Output
22	L0DAT(5)	Output enable
23	L0DAT(5)	Input
24	L0DAT(6)	Output
25	L0DAT(6)	Output enable
26	L0DAT(6)	Input
27	L0DAT(7)	Output
28	L0DAT(7)	Output enable
29	L0DAT(7)	Input
30	DT0	Output
31	DT0	Output enable
32	DT0	No function
33	TCLK0	Output
34	TCLK0	Output enable
35	TCLK0	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
36	TFS0	Output
37	TFS0	Output enable
38	TFS0	Input
39	RFS0	Output
40	RFS0	Output enable
41	RFS0	Input
42	RCLK0	Output
43	RCLK0	Output enable
44	RCLK0	Input
45	DR0	Output
46	DR0	Output enable
47	DR0	Input
48	DR1	Output
49	DR1	Output enable
50	DR1	Input
51	RCLK1	Output
52	RCLK1	Output enable
53	RCLK1	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
54	RFS1	Output
55	RFS1	Output enable
56	RFS1	Input
57	TFS1	Output
58	TFS1	Output enable
59	TFS1	Input
60	TCLK1	Output
61	TCLK1	Output enable
62	TCLK1	Input
63	DT1	Output
64	DT1	Output enable
65	DT1	No function
66	TIMEXP	Output
67	TIMEXP	No function
68	TIMEXP	No function
69	FLAG0	Output
70	FLAG0	Output enable
71	FLAG0	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
72	FLAG1	Output
73	FLAG1	Output enable
74	FLAG1	Input
75	FLAG2	Output
76	FLAG2	Output enable
77	FLAG2	Input
78	FLAG3	Output
79	FLAG3	Output enable
80	FLAG3	Input
81	IRQ0_B	No function
82	IRQ0_B	No function
83	IRQ0_B	Input
84	IRQ1_B	No function
85	IRQ1_B	No function
86	IRQ1_B	Input
87	IRQ2_B	No function
88	IRQ2_B	No function
89	IRQ2_B	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
90	RPBA	No function
91	RPBA	No function
92	RPBA	Input
93	RESET_B	No function
94	RESET_B	No function
95	RESET_B	Input
96	EMU_B	Output
97	EMU_B	Output enable
98	EMU_B	No function
99	DATA(0)	Output
100	DATA(0)	Output enable
101	DATA(0)	Input
102	DATA(1)	Output
103	DATA(1)	Output enable
104	DATA(1)	Input
105	DATA(2)	Output
106	DATA(2)	Output enable
107	DATA(2)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
108	DATA(3)	Output
109	DATA(3)	Output enable
110	DATA(3)	Input
111	DATA(4)	Output
112	DATA(4)	Output enable
113	DATA(4)	Input
114	DATA(5)	Output
115	DATA(5)	Output enable
116	DATA(5)	Input
117	DATA(6)	Output
118	DATA(6)	Output enable
119	DATA(6)	Input
120	DATA(7)	Output
121	DATA(7)	Output enable
122	DATA(7)	Input
123	DATA(8)	Output
124	DATA(8)	Output enable
125	DATA(8)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
126	DATA(9)	Output
127	DATA(9)	Output enable
128	DATA(9)	Input
129	DATA(10)	Output
130	DATA(10)	Output enable
131	DATA(10)	Input
132	DATA(11)	Output
133	DATA(11)	Output enable
134	DATA(11)	Input
135	DATA(12)	Output
136	DATA(12)	Output enable
137	DATA(12)	Input
138	DATA(13)	Output
139	DATA(13)	Output enable
140	DATA(13)	Input
141	DATA(14)	Output
142	DATA(14)	Output enable
143	DATA(14)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
144	DATA(15)	Output
145	DATA(15)	Output enable
146	DATA(15)	Input
147	DATA(16)	Output
148	DATA(16)	Output enable
149	DATA(16)	Input
150	DATA(17)	Output
151	DATA(17)	Output enable
152	DATA(17)	Input
153	DATA(18)	Output
154	DATA(18)	Output enable
155	DATA(18)	Input
156	DATA(19)	Output
157	DATA(19)	Output enable
158	DATA(19)	Input
159	DATA(20)	Output
160	DATA(20)	Output enable
161	DATA(20)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
162	DATA(21)	Output
163	DATA(21)	Output enable
164	DATA(21)	Input
165	DATA(22)	Output
166	DATA(22)	Output enable
167	DATA(22)	Input
168	DATA(23)	Output
169	DATA(23)	Output enable
170	DATA(23)	Input
171	DATA(24)	Output
172	DATA(24)	Output enable
173	DATA(24)	Input
174	DATA(25)	Output
175	DATA(25)	Output enable
176	DATA(25)	Input
177	DATA(26)	Output
178	DATA(26)	Output enable
179	DATA(26)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
180	DATA(27)	Output
181	DATA(27)	Output enable
182	DATA(27)	Input
183	DATA(28)	Output
184	DATA(28)	Output enable
185	DATA(28)	Input
186	DATA(29)	Output
187	DATA(29)	Output enable
188	DATA(29)	Input
189	DATA(30)	Output
190	DATA(30)	Output enable
191	DATA(30)	Input
192	DATA(31)	Output
193	DATA(31)	Output enable
194	DATA(31)	Input
195	DATA(32)	Output
196	DATA(32)	Output enable
197	DATA(32)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
198	DATA(33)	Output
199	DATA(33)	Output enable
200	DATA(33)	Input
201	DATA(34)	Output
202	DATA(34)	Output enable
203	DATA(34)	Input
204	DATA(35)	Output
205	DATA(35)	Output enable
206	DATA(35)	Input
207	DATA(36)	Output
208	DATA(36)	Output enable
209	DATA(36)	Input
210	DATA(37)	Output
211	DATA(37)	Output enable
212	DATA(37)	Input
213	DATA(38)	Output
214	DATA(38)	Output enable
215	DATA(38)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
216	DATA(39)	Output
217	DATA(39)	Output enable
218	DATA(39)	Input
219	DATA(40)	Output
220	DATA(40)	Output enable
221	DATA(40)	Input
222	DATA(41)	Output
223	DATA(41)	Output enable
224	DATA(41)	Input
225	DATA(42)	Output
226	DATA(42)	Output enable
227	DATA(42)	Input
228	DATA(43)	Output
229	DATA(43)	Output enable
230	DATA(43)	Input
231	DATA(44)	Output
232	DATA(44)	Output enable
233	DATA(44)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
234	DATA(45)	Output
235	DATA(45)	Output enable
236	DATA(45)	Input
237	DATA(46)	Output
238	DATA(46)	Output enable
239	DATA(46)	Input
240	DATA(47)	Output
241	DATA(47)	Output enable
242	DATA(47)	Input
243	CLK_CFG0	No function
244	CLK_CFG0	No function
245	CLK_CFG0	Input
246	CLK_CFG1	No function
247	CLK_CFG1	No function
248	CLK_CFG1	Input
249	CLK_CFG2	No function
250	CLK_CFG2	No function
251	CLK_CFG2	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
252	CLK_CFG3	No function
253	CLK_CFG3	No function
254	CLK_CFG3	Input
255	CLKOUT	Output
256	CLKOUT	Output enable
257	CLKOUT	No function
258	DATA(48)	Output
259	DATA(48)	Output enable
260	DATA(48)	Input
261	DATA(49)	Output
262	DATA(49)	Output enable
263	DATA(49)	Input
264	DATA(50)	Output
265	DATA(50)	Output enable
266	DATA(50)	Input
267	DATA(51)	Output
268	DATA(51)	Output enable
269	DATA(51)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
270	DATA(52)	Output
271	DATA(52)	Output enable
272	DATA(52)	Input
273	DATA(53)	Output
274	DATA(53)	Output enable
275	DATA(53)	Input
276	DATA(54)	Output
277	DATA(54)	Output enable
278	DATA(54)	Input
279	DATA(55)	Output
280	DATA(55)	Output enable
281	DATA(55)	Input
282	DATA(56)	Output
283	DATA(56)	Output enable
284	DATA(56)	Input
285	DATA(57)	Output
286	DATA(57)	Output enable
287	DATA(57)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
288	DATA(58)	Output
289	DATA(58)	Output enable
290	DATA(58)	Input
291	DATA(59)	Output
292	DATA(59)	Output enable
293	DATA(59)	Input
294	DATA(60)	Output
295	DATA(60)	Output enable
296	DATA(60)	Input
297	DATA(61)	Output
298	DATA(61)	Output enable
299	DATA(61)	Input
300	DATA(62)	Output
301	DATA(62)	Output enable
302	DATA(62)	Input
303	DATA(63)	Output
304	DATA(63)	Output enable
305	DATA(63)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
306	ADDR(2)	Output
307	ADDR(2)	Output enable
308	ADDR(2)	Input
309	ADDR(3)	Output
310	ADDR(3)	Output enable
311	ADDR(3)	Input
312	ADDR(4)	Output
313	ADDR(4)	Output enable
314	ADDR(4)	Input
315	ADDR(5)	Output
316	ADDR(5)	Output enable
317	ADDR(5)	Input
318	ADDR(6)	Output
319	ADDR(6)	Output enable
320	ADDR(6)	Input
321	ADDR(7)	Output
322	ADDR(7)	Output enable
323	ADDR(7)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
324	ADDR(8)	Output
325	ADDR(8)	Output enable
326	ADDR(8)	Input
327	ADDR(9)	Output
328	ADDR(9)	Output enable
329	ADDR(9)	Input
330	ADDR(10)	Output
331	ADDR(10)	Output enable
332	ADDR(10)	Input
333	ADDR(11)	Output
334	ADDR(11)	Output enable
335	ADDR(11)	Input
336	ADDR(12)	Output
337	ADDR(12)	Output enable
338	ADDR(12)	Input
339	ADDR(13)	Output
340	ADDR(13)	Output enable
341	ADDR(13)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
342	ADDR(14)	Output
343	ADDR(14)	Output enable
344	ADDR(14)	Input
345	ADDR(15)	Output
346	ADDR(15)	Output enable
347	ADDR(15)	Input
348	ADDR(16)	Output
349	ADDR(16)	Output enable
350	ADDR(16)	Input
351	ADDR(17)	Output
352	ADDR(17)	Output enable
353	ADDR(17)	Input
354	ADDR(18)	Output
355	ADDR(18)	Output enable
356	ADDR(18)	Input
357	ADDR(19)	Output
358	ADDR(19)	Output enable
359	ADDR(19)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
360	ADDR(20)	Output
361	ADDR(20)	Output enable
362	ADDR(20)	Input
363	ADDR(21)	Output
364	ADDR(21)	Output enable
365	ADDR(21)	Input
366	ADDR(22)	Output
367	ADDR(22)	Output enable
368	ADDR(22)	Input
369	ADDR(23)	Output
370	ADDR(23)	Output enable
371	ADDR(23)	Input
372	ADDR(24)	Output
373	ADDR(24)	Output enable
374	ADDR(24)	Input
375	ADDR(25)	Output
376	ADDR(25)	Output enable
377	ADDR(25)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
378	ADDR(26)	Output
379	ADDR(26)	Output enable
380	ADDR(26)	Input
381	ADDR(27)	Output
382	ADDR(27)	Output enable
383	ADDR(27)	Input
384	ADDR(28)	Output
385	ADDR(28)	Output enable
386	ADDR(28)	Input
387	ADDR(29)	Output
388	ADDR(29)	Output enable
389	ADDR(29)	Input
390	ADDR(30)	Output
391	ADDR(30)	Output enable
392	ADDR(30)	Input
393	ADDR(31)	Output
394	ADDR(31)	Output enable
395	ADDR(31)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
396	ID0	No function
397	ID0	No function
398	ID0	Input
399	ID1	No function
400	ID1	No function
401	ID1	Input
402	ID2	No function
403	ID2	No function
404	ID2	Input
405	ADDR(0)	Output
406	ADDR(0)	Output enable
407	ADDR(0)	Input
408	ADDR(1)	Output
409	ADDR(1)	Output enable
410	ADDR(1)	Input
411	BRST	Output
412	BRST	Output enable
413	BRST	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
414	BMS_B	Output
415	BMS_B	Output enable
416	BMS_B	Input
417	MS_B(0)	Output
418	MS_B(0)	Output enable
419	MS_B(0)	No function
420	MS_B(1)	Output
421	MS_B(1)	Output enable
422	MS_B(1)	No function
423	MS_B(2)	Output
424	MS_B(2)	Output enable
425	MS_B(2)	No function
426	MS_B(3)	Output
427	MS_B(3)	Output enable
428	MS_B(3)	No function
429	CIF_B	Output
430	CIF_B	Output enable
431	CIF_B	No function

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
432	CS_B	No function
433	CS_B	No function
434	CS_B	Input
435	WRH_B	Output
436	WRH_B	Output enable
437	WRH_B	Input
438	WRL_B	Output
439	WRL_B	Output enable
440	WRL_B	Input
441	RDH_B	Output
442	RDH_B	Output enable
443	RDH_B	Input
444	RDL_B	Output
445	RDL_B	Output enable
446	RDL_B	Input
447	DMAG1_B	Output
448	DMAG1_B	Output enable
449	DMAG1_B	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
450	DMAG2_B	Output
451	DMAG2_B	Output enable
452	DMAG2_B	Input
453	DMAR1_B	Output
454	DMAR1_B	Output enable
455	DMAR1_B	Input
456	DMAR2_B	Output
457	DMAR2_B	Output enable
458	DMAR2_B	Input
459	LBOOT	No function
460	LBOOT	No function
461	LBOOT	Input
462	EBOOT	No function
463	EBOOT	No function
464	EBOOT	Input
465	L5DAT(0)	Output
466	L5DAT(0)	Output enable
467	L5DAT(0)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
468	L5DAT(1)	Output
469	L5DAT(1)	Output enable
470	L5DAT(1)	Input
471	L5DAT(2)	Output
472	L5DAT(2)	Output enable
473	L5DAT(2)	Input
474	L5DAT(3)	Output
475	L5DAT(3)	Output enable
476	L5DAT(3)	Input
477	L5ACK	Output
478	L5ACK	Output enable
479	L5ACK	Input
480	L5CLK	Output
481	L5CLK	Output enable
482	L5CLK	Input
483	L5DAT(4)	Output
484	L5DAT(4)	Output enable
485	L5DAT(4)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
486	L5DAT(5)	Output
487	L5DAT(5)	Output enable
488	L5DAT(5)	Input
489	L5DAT(6)	Output
490	L5DAT(6)	Output enable
491	L5DAT(6)	Input
492	L5DAT(7)	Output
493	L5DAT(7)	Output enable
494	L5DAT(7)	Input
495	L4DAT(0)	Output
496	L4DAT(0)	Output enable
497	L4DAT(0)	Input
498	L4DAT(1)	Output
499	L4DAT(1)	Output enable
500	L4DAT(1)	Input
501	L4DAT(2)	Output
502	L4DAT(2)	Output enable
503	L4DAT(2)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
504	L4DAT(3)	Output
505	L4DAT(3)	Output enable
506	L4DAT(3)	Input
507	L4ACK	Output
508	L4ACK	Output enable
509	L4ACK	Input
510	L4CLK	Output
511	L4CLK	Output enable
512	L4CLK	Input
513	L4DAT(4)	Output
514	L4DAT(4)	Output enable
515	L4DAT(4)	Input
516	L4DAT(5)	Output
517	L4DAT(5)	Output enable
518	L4DAT(5)	Input
519	L4DAT(6)	Output
520	L4DAT(6)	Output enable
521	L4DAT(6)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
522	L4DAT(7)	Output
523	L4DAT(7)	Output enable
524	L4DAT(7)	Input
525	L3DAT(0)	Output
526	L3DAT(0)	Output enable
527	L3DAT(0)	Input
528	L3DAT(1)	Output
529	L3DAT(1)	Output enable
530	L3DAT(1)	Input
531	L3DAT(2)	Output
532	L3DAT(2)	Output enable
533	L3DAT(2)	Input
534	L3DAT(3)	Output
535	L3DAT(3)	Output enable
536	L3DAT(3)	Input
537	L3ACK	Output
538	L3ACK	Output enable
539	L3ACK	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
540	L3CLK	Output
541	L3CLK	Output enable
542	L3CLK	Input
543	L3DAT(4)	Output
544	L3DAT(4)	Output enable
545	L3DAT(4)	Input
546	L3DAT(5)	Output
547	L3DAT(5)	Output enable
548	L3DAT(5)	Input
549	L3DAT(6)	Output
550	L3DAT(6)	Output enable
551	L3DAT(6)	Input
552	L3DAT(7)	Output
553	L3DAT(7)	Output enable
554	L3DAT(7)	Input
555	SBTS_B	No function
556	SBTS_B	No function
557	SBTS_B	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
558	PAGE	Output
559	PAGE	Output enable
560	PAGE	No function
561	PA_B	Output
562	PA_B	Output enable
563	PA_B	Input
564	REDY	Output
565	REDY	Output enable
566	REDY	No function
567	ACK	Output
568	ACK	Output enable
569	ACK	Input
570	BR_B(1)	Output
571	BR_B(1)	Output enable
572	BR_B(1)	Input
573	BR_B(2)	Output
574	BR_B(2)	Output enable
575	BR_B(2)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
576	BR_B(3)	Output
577	BR_B(3)	Output enable
578	BR_B(3)	Input
579	BR_B(4)	Output
580	BR_B(4)	Output enable
581	BR_B(4)	Input
582	BR_B(5)	Output
583	BR_B(5)	Output enable
584	BR_B(5)	Input
585	BR_B(6)	Output
586	BR_B(6)	Output enable
587	BR_B(6)	Input
588	HBR_B	No function
589	HBR_B	No function
590	HBR_B	Input
591	HBG_B	Output
592	HBG_B	Output enable
593	HBG_B	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
594	L2DAT(0)	Output
595	L2DAT(0)	Output enable
596	L2DAT(0)	Input
597	L2DAT(1)	Output
598	L2DAT(1)	Output enable
599	L2DAT(1)	Input
600	L2DAT(2)	Output
601	L2DAT(2)	Output enable
602	L2DAT(2)	Input
603	L2DAT(3)	Output
604	L2DAT(3)	Output enable
605	L2DAT(3)	Input
606	L2ACK	Output
607	L2ACK	Output enable
608	L2ACK	Input
609	L2CLK	Output
610	L2CLK	Output enable
611	L2CLK	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
612	L2DAT(4)	Output
613	L2DAT(4)	Output enable
614	L2DAT(4)	Input
615	L2DAT(5)	Output
616	L2DAT(5)	Output enable
617	L2DAT(5)	Input
618	L2DAT(6)	Output
619	L2DAT(6)	Output enable
620	L2DAT(6)	Input
621	L2DAT(7)	Output
622	L2DAT(7)	Output enable
623	L2DAT(7)	Input
624	L1DAT(0)	Output
625	L1DAT(0)	Output enable
626	L1DAT(0)	Input
627	L1DAT(1)	Output
628	L1DAT(1)	Output enable
629	L1DAT(1)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
630	L1DAT(2)	Output
631	L1DAT(2)	Output enable
632	L1DAT(2)	Input
633	L1DAT(3)	Output
634	L1DAT(3)	Output enable
635	L1DAT(3)	Input
636	L1ACK	Output
637	L1ACK	Output enable
638	L1ACK	Input
639	L1CLK	Output
640	L1CLK	Output enable
641	L1CLK	Input
642	L1DAT(4)	Output
643	L1DAT(4)	Output enable
644	L1DAT(4)	Input
645	L1DAT(5)	Output
646	L1DAT(5)	Output enable
647	L1DAT(5)	Input

Table 10-7. JTAG Boundary Register (Cont'd)

Scan #	Signal Name	Latch Type
648	L1DAT(6)	Output
649	L1DAT(6)	Output enable
650	L1DAT(6)	Input
651	L1DAT(7)	Output
652	L1DAT(7)	Output enable
653	L1DAT(7)	Input
654	SPARE	No function

Device Identification Register

No device identification register is included in the ADSP-21160 DSP.

Built-in Self-test Operation (BIST)

No self-test functions are supported by the ADSP-21160 DSP.

Private Instructions

Table 10-2 on page 10-4 lists the private instructions that are reserved for emulation and memory test. The ADSP-21160 EZ-ICE emulator uses the TAP and boundary scan as a way to access the processor in the target system. The EZ-ICE emulator requires a target board connector for access to

the TAP. For more information, see “Designing for JTAG Emulation” on page 11-24.

References

- IEEE Standard 1149.1-1990. Standard Test Access Port and Boundary-Scan Architecture.

  To order a copy, contact IEEE at 1-800-678-IEEE.

- Maunder, C.M. and R. Tulloss. Test Access Ports and Boundary Scan Architectures.

  IEEE Computer Society Press, 1991.

- Parker, Kenneth. The Boundary Scan Handbook.

  Kluwer Academic Press, 1992.

- Bleeker, Harry, P. van den Eijnden, and F. de Jong. Boundary-Scan Test—A Practical Approach.

  Kluwer Academic Press, 1993.

- Hewlett-Packard Co. HP Boundary-Scan Tutorial and BSDL Reference Guide.

  (HP part# E1017-90001.) 1992.

11 SYSTEM DESIGN

The DSP supports many system design options. The options implemented in a system are influenced by cost, performance, and system requirements. This chapter provides the following system design information:

- "DSP Pin Descriptions" on page 11-1
- "Designing for JTAG Emulation" on page 11-24
- "Conditioning Input Signals" on page 11-35
- "Designing For High Frequency Operation" on page 11-36
- "Booting Single and Multiple Processors" on page 11-48

Other chapters also discuss system design issues. Some other locations for system design information include:

- "Setting External Port Modes" on page 7-2
- "Setting Link Port Modes" on page 8-5
- "Setting Serial Port Modes" on page 9-7

DSP Pin Descriptions

This section describes the pins of the DSP and shows how these signals can be used in a DSP system. Figure 11-1 illustrates how the pins are used

in a single-processor system. Figure 7-27 on page 7-96 shows a system diagram illustrating pin connections in an DSP multiprocessor cluster.

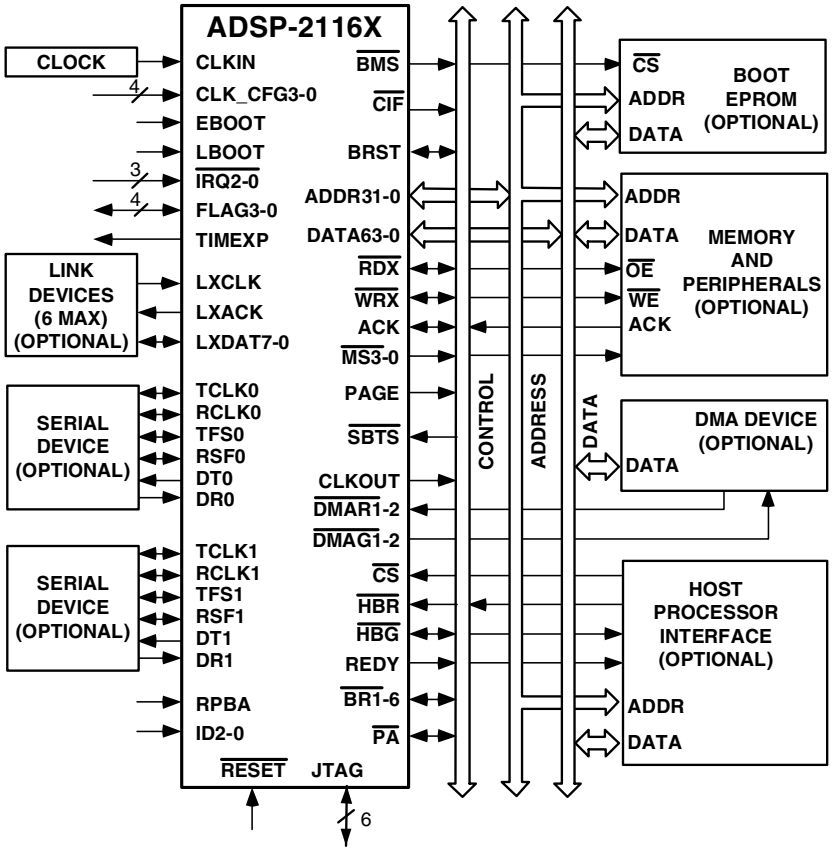


Figure 11-1. Single-Processor DSP System

DSP pin definitions are listed in Table 11-1. The following symbols appear in the **Type** column of Table 11-1:

Table 11-1. Pin Descriptions

Pin	Type	Function
ADDR31-0	I/O/T	**External Bus Address**. The DSP outputs addresses for external memory and peripherals on these pins. In a multiprocessor system, the bus master outputs addresses for read/writes of the internal memory or I/O processor registers of other DSPs. The DSP inputs addresses when a host processor or multiprocessing bus master is reading or writing its internal memory or I/O processor registers. A keeper latch on the DSP's ADDR31-0 pins maintains the input at the level it was last driven (only enabled on the DSP with ID2-0=00x).
DATA63-0	I/O/T	**External Bus Data**. The DSP inputs and outputs data and instructions on these pins. Pull-up resistors on unused DATA pins are not necessary. A keeper latch on the DSP's DATA63-0 pins maintains the input at the level it was last driven (only enabled on the DSP with ID2-0=00x).
$\overline{\text{MS3-0}}$	O/T	**Memory Select Lines**. These outputs are asserted (low) as chip selects for the corresponding banks of external memory. Memory bank size must be defined in the SYSCON control register. The $\overline{\text{MS3-0}}$ outputs are decoded memory address lines. In asynchronous access mode, the $\overline{\text{MS3-0}}$ outputs transition with the other address outputs. In synchronous access modes, the $\overline{\text{MS3-0}}$ outputs assert with the other address lines. They deassert after the first CLKIN cycle in which ACK is sampled asserted. $\overline{\text{MS3-0}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
$\overline{\text{CIF}}$	O	**Core Instruction Fetch.** Signal is active low when an external instruction fetch is performed. Driven by bus master only. Three-state when host is bus master. $\overline{\text{CIF}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
$\overline{\text{RDL}}$	I/O/T	**Memory Read Low Strobe.** $\overline{\text{RDL}}$ is asserted whenever DSP reads from the low word of external memory or from the internal memory of other DSPs. External devices, including other DSPs, must assert $\overline{\text{RDL}}$ for reading from low word of DSP internal memory. In a multiprocessing system, $\overline{\text{RDL}}$ is driven by the bus master $\overline{\text{RDL}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
$\overline{\text{RDH}}$	I/O/T	**Memory Read High Strobe.** $\overline{\text{RDH}}$ is asserted whenever DSP reads from the high word of external memory or from the internal memory of other DSPs. External devices, including other DSPs, must assert $\overline{\text{RDH}}$ for reading from the high word of DSP internal memory. In a multiprocessing system, $\overline{\text{RDH}}$ is driven by the bus master. $\overline{\text{RDH}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
$\overline{\text{WRL}}$	I/O/T	**Memory Write Low Strobe.** $\overline{\text{WRL}}$ is asserted when DSP writes to the low word of external memory or internal memory of other DSPs. External devices must assert $\overline{\text{WRL}}$ for writing to DSP's low word of internal memory. In a multiprocessing system, $\overline{\text{WRL}}$ is driven by the bus master. $\overline{\text{WRL}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
$\overline{\text{WRH}}$	I/O/T	**Memory Write High Strobe.** $\overline{\text{WRH}}$ is asserted when DSP writes to the high word of external memory or internal memory of other DSPs. External devices must assert $\overline{\text{WRH}}$ for writing to DSP's high word of internal memory. In a multiprocessing system, $\overline{\text{WRH}}$ is driven by the bus master. $\overline{\text{WRH}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
BRST	I/O/T	**Sequential burst access.** BRST is asserted by DSP or a host to indicate that data associated with consecutive addresses is being read or written. A slave device samples the initial address and increments an internal address counter after each transfer. The incremented address is not pipelined on the bus. If the burst access is a read from the DSP by a host, DSP increments the address automatically as long as BRST is asserted. BRST is asserted after the initial access of a burst transfer. It is asserted for every cycle after that, except for the last data request cycle (denoted by $\overline{\text{RDx}}$ or $\overline{\text{WRx}}$ asserted and BRST negated). A keeper latch on the DSP's BRST pin maintains the input at the level it was last driven (only enabled on the DSP with ID2-0=00x).
PAGE	O/T	**DRAM Page Boundary**. The DSP asserts this pin to signal that an external DRAM page boundary has been crossed. DRAM page size must be defined in the DSP's memory control register (WAIT). DRAM can only be implemented in external memory Bank 0. The PAGE signal can only be activated for Bank 0 accesses. In a multiprocessing system PAGE is output by the bus master. A keeper latch on the DSP's PAGE pin maintains the output at the level it was last driven (only enabled on the DSP with ID2-0=00x).

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
ACK	I/O/S	**Memory Acknowledge**. External devices can deassert ACK (low) to add wait states to an external memory access. ACK is used by I/O devices, memory controllers, or other peripherals to hold off completion of an external memory access. The DSP deasserts ACK as an output to add wait states to a synchronous access of its internal memory. ACK has a 2kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x
$\overline{\text{SBTS}}$	I/S	**Suspend Bus and Three-State**. External devices can assert $\overline{\text{SBTS}}$ (low) to place the external bus address, data, selects, and strobes in a high impedance state for the following cycle. If the DSP attempts to access external memory while $\overline{\text{SBTS}}$ is asserted, the processor halts and the memory access does not complete until $\overline{\text{SBTS}}$ is deasserted. $\overline{\text{SBTS}}$ should only be used to recover from host processor/DSP deadlock or used with a DRAM controller.
$\overline{\text{IRQ2-0}}$	I/A	**Interrupt Request Lines**. These are sampled on the rising edge of CLKIN and may be either edge-triggered or level-sensitive.
FLAG3-0	I/O/A	**Flag Pins**. Each is configured via control bits as either an input or output. As an input, it can be tested as a condition. As an output, it can be used to signal external peripherals.
TIMEXP	O	**Timer Expired**. Asserted for four core clock cycles when the timer is enabled, and TCOUNT decrements to zero.
$\overline{\text{HBR}}$	I/A	**Host Bus Request**. Must be asserted by a host processor to request control of the DSP's external bus. When $\overline{\text{HBR}}$ is asserted in a multiprocessing system, the DSP that is bus master relinquishes the bus and asserts $\overline{\text{HBG}}$. To relinquish the bus, the DSP places the address, data, select, and strobe lines in a high impedance state. $\overline{\text{HBR}}$ has priority over all DSP bus requests ($\overline{\text{BR6-1}}$) in a multiprocessing system.
$\overline{\text{HBG}}$	I/O	**Host Bus Grant**. Acknowledges an $\overline{\text{HBR}}$ bus request, indicating that the host processor may take control of the external bus. $\overline{\text{HBG}}$ is asserted (held low) by the ADSP-21535 until $\overline{\text{HBR}}$ is released. In a multiprocessing system, $\overline{\text{HBG}}$ is output by the DSP bus master and is monitored by all others. After $\overline{\text{HBR}}$ is asserted and before $\overline{\text{HBG}}$ is deasserted, $\overline{\text{HBG}}$ will float for one CLKIN cycle. To avoid erroneous grants, $\overline{\text{HBG}}$ should be pulled up with a 20kΩ to 50kΩ external resistor.

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
$\overline{\text{CS}}$	I/A	**Chip Select**. Asserted by host processor to select the DSP.
REDY	O (O/D)	**Host Bus Acknowledge**. The DSP deasserts REDY (low) to add wait-states to a host access when $\overline{\text{CS}}$ and $\overline{\text{HBR}}$ inputs are asserted.
$\overline{\text{DMAR1}}$	I/A	**DMA Request 1** (DMA Channel 11). Asserted by external port devices to request DMA services. $\overline{\text{DMAR1}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x.
$\overline{\text{DMAR2}}$	I/A	**DMA Request 2** (DMA Channel 12). Asserted by external port devices to request DMA services. $\overline{\text{DMAR2}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x
$\overline{\text{DMAG1}}$	O/T	**DMA Grant 1** (DMA Channel 11). Asserted by DSP to indicate that the requested DMA starts on the next cycle. Driven by bus master only. $\overline{\text{DMAG1}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x
$\overline{\text{DMAG2}}$	O/T	**DMA Grant 2** (DMA Channel 12). Asserted by DSP to indicate that the requested DMA starts on the next cycle. Driven by bus master only. $\overline{\text{DMAG2}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x
$\overline{\text{BR6-1}}$	I/O/S	**Multiprocessing Bus Requests**. Used by multiprocessing DSPs to arbitrate for bus mastership. Each DSP only drives its own $\overline{\text{BRx}}$ line (corresponding to the value of its ID2-0 inputs) and monitors all others. In a multiprocessor system with less than six DSPs, the unused $\overline{\text{BRx}}$ pins should be pulled high. The processor's own $\overline{\text{BRx}}$ line must not be pulled high or low because it is an output.
ID2-0	I	**Multiprocessing ID**. Determines which multiprocessing bus request ($\overline{\text{BR1}}$-$\overline{\text{BR6}}$) is used by each DSP. ID = 001 corresponds to $\overline{\text{BR1}}$, ID = 010 corresponds to $\overline{\text{BR2}}$, and so on. Use ID = 000 in single-processor systems. These lines are a system configuration selection which should be hardwired or only changed at reset.
RPBA	I/S	**Rotating Priority Bus Arbitration Select**. When RPBA is high, rotating priority for multiprocessor bus arbitration is selected. When RPBA is low, fixed priority is selected. This signal is a system configuration selection that must be set to the same value on every DSP. If the value of RPBA is changed during system operation, it must be changed in the same CLKIN cycle on every DSP.

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
$\overline{\text{PA}}$	I/O/T	**Priority Access**. Asserting its $\overline{\text{PA}}$ pin allows an DSP bus slave to interrupt background DMA transfers and gain access to the external bus. $\overline{\text{PA}}$ is connected to all DSPs in the system. If access priority is not required in a system, the $\overline{\text{PA}}$ pin should be left unconnected. $\overline{\text{DMAR1}}$ has a 20kΩ internal pull up resistor that is enabled on the DSP with ID2-0=00x
DTx	O	**Data Transmit** (Serial Ports 0, 1). Each DT pin has a 50kΩ internal pull-up resistor.
DRx	I	**Data Receive** (Serial Ports 0, 1). Each DR pin has a 50kΩ internal pull-up resistor.
TCLKx	I/O	**Transmit Clock** (Serial Ports 0, 1). Each TCLK pin has a 50kΩ internal pull-up resistor.
RCLKx	I/O	**Receive Clock** (Serial Ports 0, 1). Each RCLK pin has a 50kΩ internal pull-up resistor.
TFSx	I/O	**Transmit Frame Sync** (Serial Ports 0, 1).
RFSx	I/O	**Receive Frame Sync** (Serial Ports 0, 1).
LxDAT7-0	I/O	**Link Port Data** (Link Ports 0-5). Each LxDAT pin has an internal pull-down resistor that is enabled or disabled by the LPDRD bit of the LCTL0-1 register.
LxCLK	I/O	**Link Port Clock** (Link Ports 0-5). Each LxCLK pin has an internal pull-down resistor that is enabled or disabled by the LPDRD bit of the LCTL0-1 register.
LxACK	I/O	**Link Port Acknowledge** (Link Ports 0-5). Each LxACK pin has an internal pull-down resistor that is enabled or disabled by the LPDRD bit of the LCTL0-1 register.
EBOOT	I	**EPROM Boot Select**. For a description of how this pin operates, see the $\overline{\text{BMS}}$ pin description. This signal is a system configuration selection that should be hardwired.
LBOOT	I	**Link Boot**. For a description of how this pin operates, see the $\overline{\text{BMS}}$ pin description. This signal is a system configuration selection that should be hardwired.

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
$\overline{\text{BMS}}$	I/O/T	**Boot Memory Select**. Serves as an output or input as selected with the EBOOT and LBOOT pins as shown below. This input is a system configuration selection that should be hardwired. **EBOOTLBOOT$\overline{\text{BMS}}$Booting Mode** 10OutputEPROM (Connect $\overline{\text{BMS}}$ to EPROM chip select.) 001 (Input)Host Processor 011 (Input)Link Port 000 (Input)No Booting. Processor executes from external memory 010 (Input)Reserved 11x (Input)Reserved
CLKIN	I	**Local Clock In.** CLKIN is the DSP clock input. The DSP external port cycles at the frequency of CLKIN. The CLKIN frequency is multiplied by a ratio (CLK_CFG3-0) to select the instruction cycle rate, which is programmable at powerup. CLKIN may not be halted, changed, or operated below the specified frequency.
CLK_ CFG3-0	I	**Core/CLKIN Ratio Control.** DSP core clock (instruction cycle) rate is equal to n x CLKIN where n is user selectable to 2, 3, or 4, using the CLK_CFG3-0 inputs.
CLKOUT	O/T	**Local Clock Out.** CLKOUT is driven at the CLKIN frequency by the DSP. This output is three-stated by setting the COD bit in the SYSCON register. A keeper latch on the DSP's CLKOUT pin maintains the output at the level it was last driven (only enabled on the DSP with ID2-0=00x).
$\overline{\text{RESET}}$	I/A	**Processor Reset**. Resets the DSP to a known state and begins execution at the program memory location specified by the hardware reset vector address. The $\overline{\text{RESET}}$ input must be asserted (low) at power-up. The only difference between the soft and hard reset is that the external bus arbitration is not affected by a soft reset. The PLL does not get reset by a soft reset.
TCK	I	**Test Clock (JTAG)**. Provides a clock for JTAG boundary scan.
TMS	I/S	**Test Mode Select (JTAG)**. Used to control the test state machine. TMS has an internal pull-up resistor.
TDI	I/S	**Test Data Input (JTAG)**. Provides serial data for the boundary scan logic. TDI has an internal pull-up resistor.

Table 11-1. Pin Descriptions (Cont'd)

Pin	Type	Function
TDO	O	**Test Data Output (JTAG)**. Serial scan output of the boundary scan path.
$\overline{\text{TRST}}$	I/A	**Test Reset (JTAG)**. Resets the test state machine. $\overline{\text{TRST}}$ must be asserted (pulsed low) after power-up or held low for proper operation of the DSP. $\overline{\text{TRST}}$ has an internal pull-up resistor.
$\overline{\text{EMU}}$	O (o/d)	**Emulation Status**. Must be connected to the DSP target board connector only.
VDDINT	P	**Processor Core Power Supply.** Nominally +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP). (40 pins).
VDDEXT	P	**I/O Power Supply**; Nominally +3.3V DC. (43 pins).
AVDD	P	**Analog Power Supply**; Nominally +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP). It supplies the DSP's internal PLL (clock generator). This pin has the same specifications as VDDINT, except that added filtering circuitry is required. For more information on supply specifications, see the *ADSP-21160 DSP Microcomputer Data Sheet*.
AGND	G	**Analog Power Supply Return**.
GND	G	**Power Supply Return**. (82 pins).
NC		**Do Not Connect**. Reserved pins which must be left open and unconnected. (9 pins).

A	Asynchronous
G	Ground
I	Input
O	Output
P	Power Supply
S	Synchronous
(a/d)	Active Drive

(o/d)	Open Drain
T	Three-State (when SBTS is asserted or DSP is bus slave)

Figure 11-2 shows how different data word sizes are transferred over the external port.

Inputs identified as synchronous (S) must meet timing requirements with respect to CLKIN (or with respect to TCK for TMS, TDI). Inputs identified as asynchronous (A) can be asserted asynchronously to CLKIN (or to TCK for $\overline{\text{TRST}}$).

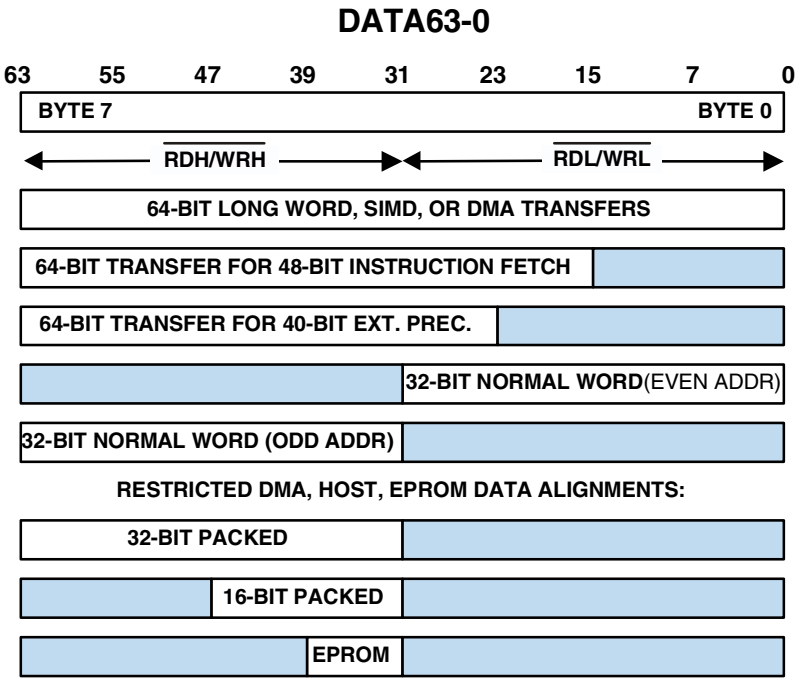


Figure 11-2. External Port Data Alignment

Unused inputs should be tied or pulled to VDD or GND, except for the following:

- ADDR31-0, DATA63-0, BRST, PAGE, CLKOUT —These pins have a logic-level hold circuit enabled on the DSP with ID2-0=00x that prevents input from floating internally.
- $\overline{\text{PA}}$, ACK, MS3-0, $\overline{\text{CIF}}$, $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, $\overline{\text{DMARx}}$, $\overline{\text{DMAGx}}$— These pins have a pull-up resistor enabled on the DSP with ID2-0=00x.
- LxDAT7-0 (LxPRDE=0), LxCLK, LxACK—These pins have an internal pull-down resistor that is controlled by bit settings in the LCTLx register.
- DTx, DRx, TCLKx, RCLKx, $\overline{\text{EMU}}$—These pins have a 50kΩ pull-up resistor.
- TMS, $\overline{\text{TRST}}$, TDI—These pins have a 20kΩ pull-up resistor.

The $\overline{\text{TRST}}$ input of the JTAG interface must be asserted (pulsed low) or held low after power-up for proper operation of the DSP. Do not leave this pin unconnected.

Additional Notes:

- In single-processor systems, the DSP owns the external bus during reset and does not perform bus arbitration to gain control of the bus.
- Operation of the $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ signals changes when $\overline{\text{CS}}$ is asserted by a host processor. For more information, see "Asynchronous Transfers" on page -59 and "Synchronous Transfers" on page -64.
- Except during a Host Transition Cycle (HTC), the $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes should not be deasserted (low-to-high transition) while ACK or $\overline{\text{REDY}}$ are deasserted (low)—the DSP hangs if this happens.

- In multiprocessor systems, the ACK signal is an input to the DSP bus master and does not float when it is not being driven, because the bus master maintains a pull-up on the pin. During reset, the ACK pin is pulled high internally with a 2 kΩ equivalent resistor by the DSP bus master and is held high with the internal pull-up resistor. It is not necessary to use an external pull-up resistor on the ACK line during booting or at any other time.
- For multiprocessor systems, PAGE is guaranteed to be asserted for the first true access after acquiring bus mastership. PAGE is not updated or asserted for multiprocessor memory space accesses or external memory space accesses to any bank other than Bank 0.
- The $\overline{\text{HBR}}$ input is disabled during any access in which the PAGE signal is asserted. This prevents the possibility of the DSP becoming a bus slave while a DRAM controller is servicing a page change.

Pin States At Reset

Table 11-2 shows the DSP pin states during and after reset.

Table 11-2. Post $\overline{\text{RESET}}$ Pin States

Pin	Type	State During and After $\overline{\text{RESET}}$
ADDR31-0	I/O/T	Driven[1]
$\overline{\text{MS3-0}}$	O/T	Driven High[1]
$\overline{\text{CIF}}$	O/T	Driven High[1]
$\overline{\text{RDH}}$	I/O/T	Driven High[1]
$\overline{\text{RDL}}$	I/O/T	Driven High[1]
$\overline{\text{WRH}}$	I/O/T	Driven High[1]
$\overline{\text{WRL}}$	I/O/T	Driven High[1]
BRST	I/O/T	Driven Low[1]

Table 11-2. Post $\overline{\text{RESET}}$ Pin States (Cont'd)

Pin	Type	State During and After $\overline{\text{RESET}}$
PAGE	O/T	Driven Low[1]
CLKOUT	O/T	Driven
ACK	I/O/S/T	Pulled High by Bus Master (w/ 2 kΩ internal pull-up resistor)[1]
$\overline{\text{HBG}}$	I/O/S/T	Driven High[1]
$\overline{\text{DMAG1}}$	O/T	Driven High[1]
$\overline{\text{DMAG2}}$	O/T	Driven High
$\overline{\text{BR6-1}}$	I/O	$\overline{\text{BR1}}$ Driven Low if Bus Master, Otherwise Driven High1
DATA63-0	I/O/T	Three-state[1]
$\overline{\text{SBTS}}$	I/S	Input; causes the master to three-state during reset[2]
$\overline{\text{IRQ2-0}}$	I/A	Inputs[2]
FLAG3-0	I/O/A	Inputs[2]
TIMEXP	O	Driven Low[2]
$\overline{\text{HBR}}$	I/A	Input[2]
$\overline{\text{CS}}$	I	Input[2]
REDY (o/d)	O	Three-state[2]
$\overline{\text{DMAR1}}$	I	Input[2]
$\overline{\text{DMAR2}}$	I	Input[2]
ID2-0	I	Inputs[2]
$\overline{\text{RPBA}}$	I/S	Input[2]
$\overline{\text{PA}}$ (o/d)	I/O	Three-state[2]
EBOOT	I	Input[2]
LBOOT	I	Input[2]
$\overline{\text{BMS}}$	I/O/T	Input[2]

Table 11-2. Post $\overline{\text{RESET}}$ Pin States (Cont'd)

Pin	Type	State During and After $\overline{\text{RESET}}$
CLKIN	I	Input
$\overline{\text{RESET}}$	I/A	Input[2]
DTx	O	Three-state (for multichannel)[3]
DRx	I	Input[3]
TCLKx	I/O	Three-state[3]
RCLKx	I/O	Three-state[3]
TFSx	I/O	Three-state[3]
RFSx	I/O	Three-state[3]
LxDAT7-0	I/O	Three-state[3]
LxCLK	I/O	Three-state[3]
LxACK	I/O	Three-state[3]
TCK	I	Input[4]
TMS	I/S	Input[4]
TDI	I/S	Input[4]
TDO	O	Three-state[4]
$\overline{\text{TRST}}$	I/A	Input[4]
$\overline{\text{EMU}}$	O (o/d)	Three-state[4]

1 Driven only by DSP bus master, otherwise three-stated
2 Bus master independent
3 Serial ports and link ports
4 JTAG interface

Clock Derivation

The DSP employs a phase-locked loop on-chip, to provide clocks that switch at higher frequencies than the system clock (CLKIN). The PLL-based

clocking methodology employed on the DSP influences the clock frequencies and behavior for the serial, link, and external ports; in addition to the processor core and internal memory. In each case, the DSP PLL provides a de-skewed clock to the port logic and I/O pins.

For the external port, this clock is fedback to the PLL, such that the external port clock always switches at the CLKIN frequency. The PLL provides internal clocks that switch at multiples of the CLKIN frequency for the internal memory, processor core, link and serial ports, and to modify the external port timing as required (for example, read/write strobes in asynchronous modes). The ratio of processor core clock frequency and CLKIN/external port clock frequency is determined by the CLK_CFG3-0 pins (as shown in Table 11-3 on page 11-17), during reset.



The core clock ratio cannot be altered dynamically. The DSP must be reset to alter the clock ratio.

The PLL provides a clock that switches at the processor core frequency to the serial and link ports. Each of the serial and link ports can be programmed to operate at clock frequencies derived from this clock. The two serial ports' transmit and receive clocks are divided down from the processor core clock frequency by setting the TDIVx and RDIVx registers appropriately. For more information, see "SPORT Transmit Divisor Registers (TDIVx)" on page -77 and "SPORT Receive Divisor Registers (RDIVx)" on page -78.

Each of the six link port clock frequencies are determined by programming the LxCLKDx parameters in the LCTL registers. For more information, see "Link Port Buffer Control Registers (LCTLx)" on page -62.

The following shows an example clock derivation:

Definition of terms:

t_{CK} = CLKIN clock period (and external port clock period)
t_{CCLK} = (processor) core clock period
t_{LCLK} = link port clock period
t_{SCLK} = serial port clock period

Clock ratios:

c_{RTO} = core/CLKIN ratio, (2, 3, or 4:1, determined by CLK_CFG)
l_{RTO} = lport/core clock ratio (1:4, 1:2, 1:3, or 1:1, determined by LxCLKD)
s_{RTO} = sport/core clock ratio (wide range determined by xCLKDIV)

Determining clock period:

$t_{CCLK} = (t_{CK}) / c_{RTO}$
$t_{LCLK} = (t_{CCLK}) * l_{RTO}$
$t_{SCLK} = (t_{CCLK}) * s_{RTO}$

RESET and CLKIN

The DSP receives its clock input on the CLKIN pin. The processor uses an on-chip phase-locked loop to generate its internal clock, which is a multiple of the CLKIN frequency. Because the phase-locked loop requires some time to achieve phase lock, CLKIN must be valid for a minimum time period during reset before the $\overline{\text{RESET}}$ signal can be deasserted. For information on minimum clock setup, see the DSP data sheet.

Table 11-3 describes the internal clock to CLKIN frequency ratios supported by DSP:

Table 11-3. Clock Configuration Definition

CLK_CFG3-0	Core/CLKIN ratio
0010	2:1
0011	3:1
0100	4:1
1111	Reserved
All others	Reserved

Table 11-4 demonstrates the internal core clock switching frequency, across a range of CLKIN (external port bus) frequencies. The minimum operational range for any given frequency is constrained by the operating range of the phase lock loop. Note that the goal in selecting a particular clock ratio for the DSP application is to provide the highest internal frequency, given a CLKIN frequency.

Table 11-4. Selecting Core to CLKIN Ratio

	CLKIN Input (MHz) ⇓			
	25	33.3	40	50
Clock Ratios	Core CLK (MHz) ⇓			
2:1	50	66.6	80	80 (21160M) 100 (21160N)
3:1	75	100	N/A	N/A
4:1	100	N/A	N/A	N/A

Input Synchronization Delay

The DSP has several asynchronous inputs: $\overline{\text{RESET}}$, $\overline{\text{TRST}}$, $\overline{\text{HBR}}$, $\overline{\text{CS}}$, $\overline{\text{DMAR1}}$, $\overline{\text{DMAR2}}$, $\overline{\text{IRQ2-0}}$, and FLAG3-0 (when configured as inputs). These inputs can

be asserted in arbitrary phase to the processor clock, CLKIN. The DSP synchronizes the inputs prior to recognizing them. The delay associated with recognition is called the synchronization delay.

Any asynchronous input must be valid prior to the recognition point to be recognized in a particular cycle. If an input does not meet the setup time on a given cycle, it may be recognized in the current cycle or during the next cycle.

To ensure recognition of an asynchronous input, it must be asserted for at least one full processor cycle plus setup and hold time, except for RESET, which must be asserted for at least four processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the DSP data sheet.

Interrupt and Timer Pins

The DSP's external interrupt pins, flag pins, and timer pin can be used to send and receive control signals to and from other devices in the system. Hardware interrupt signals are received on the IRQ2-0 pins. Interrupts can come from devices that require the DSP to perform some task on demand. A memory-mapped peripheral, for example, can use an interrupt to alert the processor that it has data available. For more information, see "Interrupts and Sequencing" on page -33.

The TIMEXP output is generated by the on-chip timer. It indicates to other devices that the programmed time period has expired. For more information, see "Timer and Sequencing" on page -50.

Flag Pins

The FLAG3-0 pins allow single-bit signalling between the DSP and other devices. For example, the DSP can raise an output flag to interrupt a host processor. Each flag pin can be programmed to be either an input or output. In addition, many DSP instructions can be conditioned on a flag's

input value, enabling efficient communication and synchronization between multiple processors or other interfaces.

The flags are bidirectional pins, each with the same functionality. The FLGxO bits in the MODE2 register program the direction of each flag pin. For more information, see "Mode Control 2 Register (MODE2)" on page A-6.

Flag Inputs

When a flag pin is programmed as an input, its value is stored in a bit in the FLAGS register. The bit is updated in each cycle with the input value from the pin. Flag inputs can be asynchronous to the DSP clock, so there is a one-cycle delay before a change on the pin appears in FLAGS (if the rising edge of the input misses the setup requirement for that cycle). For more information, see "Flag Value Register (FLAGS)" on page A-27.

An flag bit is read-only if the flag is configured as an input. Otherwise, the bit is readable and writable. The flag bit states are conditions that code can specify in conditional instructions.

Flag Outputs

When a flag is configured as an output, the value on the pin follows the value of the corresponding bit in the FLAGS register. A program can set or clear the flag bit to provide a signal to another processor or peripheral.

The FLAG outputs transition on rising edge of CLKIN. Because the processor core operates at least twice the frequency of CLKIN, the programmer must hold the FLAG state stable for at least one full CLKIN period, to insure that the output changes state. Figure 11-3 describes the relationship between instruction execution and a Flag pin, when the processor core to bus clock

ratio is set to 2:1. Figure 11-3 also describes the flag in/out process. Note that at least two instructions execute each CLKIN cycle.

```
bit set MODE2 FLG0;       /* 1st cycle: set FLAG0 to output in Mode2 */
bit clr FLAGS FLG0;       /* clear FLAG0  */
bit set FLAGS FLG0;       /* 1st cycle: set FLAG0 output high */
nop;                      /* 2nd cycle: FLAG register updated here */
                          /* A NOP indicates a NOP or another instruction
                             not related to FLAG. */
bit clr FLAGS FLG0;       /* 2nd cycle: clear FLAG0 output */
                          /* earliest assertion of FLAG0 output, depends on CLKIN phase*/
bit clr MODE2 FLG0;       /* 3rd cycle: set FLAG0 back to input */
Nop;                      /* 3rd cycle: */
nop;                      /* 4th cycle: earliest deassertion of FLAG0 output */
```

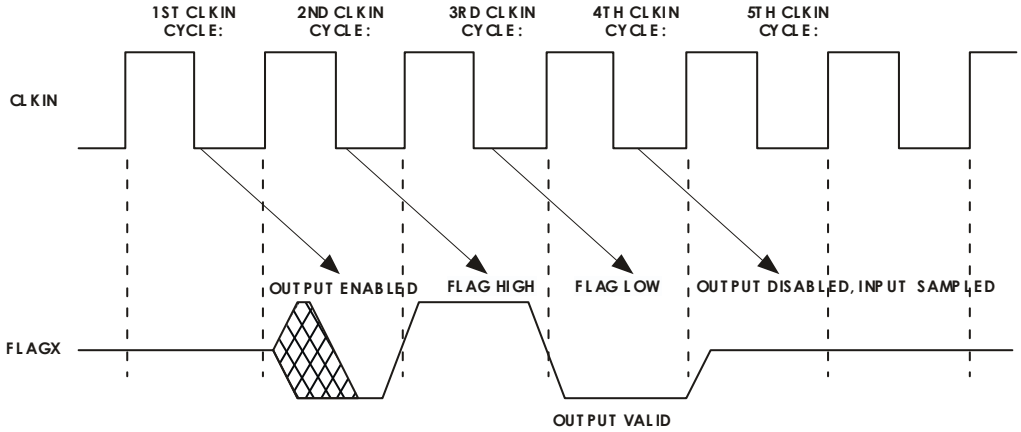


Figure 11-3. Flag Timing (At 2:1 Clock Ratio)

JTAG Interface Pins

The JTAG test access port consists of the TCK, TMS, TDI, TDO, and $\overline{\text{TRST}}$ pins. The JTAG port can be connected to a controller that performs a boundary scan for testing purposes. This port is also used by the Analog Devices (ADI) family of emulators to access on-chip emulation features. To allow the use of the emulator, a connector for its in-circuit probe must be included in the target system. For more information, see “Designing for JTAG Emulation” on page 11-24.

If $\overline{\texttt{TRST}}$ is not asserted (or held low) at power-up, the JTAG port is in an undefined state that may cause the DSP to drive out on I/O pins that would normally be three-stated at reset. $\overline{\texttt{TRST}}$ can be held low with a jumper to ground on the target board connector. For more information, see Figure 11-9 on page 11-30.

Dual-Voltage Power-up Sequencing

The ADSP-21160 dual-voltage processor has special considerations related to power-up. Note that these are general recommendations, and specifics details on dual voltage power supply systems is beyond the scope of this book. When the system power is activated through the DSP's dual power supply system, both supplies should be brought up as quickly as possible. Ideally, the two supplies, VDDEXT and VDDINT should be powered up simultaneously. Many commercially available dual supply regulators address simultaneous power-up requirements of the core and I/O. When designing a dual supply system, the designer should consider the relative voltage and ramp-up timing between the core and I/O voltages in order to avoid potential issues with system and DSP long-term reliability.

The ADSP-21160 DSP's I/O pads have a network of internal diodes to protect the DSP from damage by electrostatic discharge. These protection diodes connect the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) core and 3.3V I/O supplies internally. Figure 11-4 shows how a network of protection diodes isolates the internal supplies and provides ESD protection for the I/O pins.

During the power-up sequence of the DSP, differences in the ramp up rates and activation time between the two supplies can cause current to flow in the I/O ESD protection circuitry. When applying power separately to the VDDINT or VDDEXT pins, take precautions to prevent or limit the maximum current and duration conducted through the isolation diodes if the unpowered pins are at ground potential.

Since the ESD protection diodes connect the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) core and 3.3V I/O supplies internally, these diodes can be damaged at any time the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) core supply voltage is present without the presence of the 3.3V I/O supply.

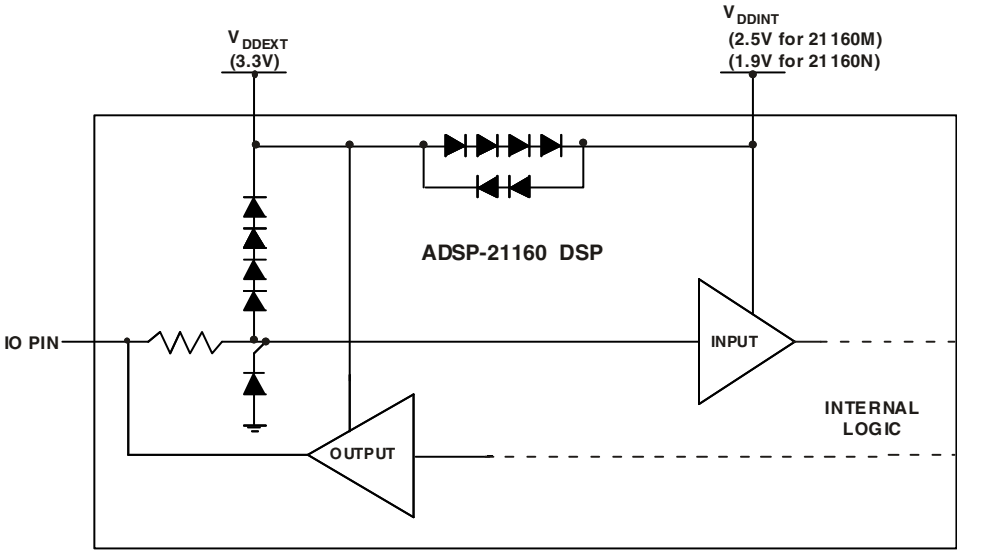


Figure 11-4. Protection Diodes and IO Pin ESD Protection

The ESD protection diodes connect the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) core and the 3.3V I/O supplies internally. Improper supply sequencing can cause damage to the ESD protection circuitry. If the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) supply is active for prolonged periods of time before the 3.3V I/O supply is activated, there is a significant amount of loading on the I/O pins. Damage occurs because the I/O will be powered from the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) supply through the ESD diodes.

To prevent this damage to the ESD diode protection circuitry, Analog Devices recommends including a bootstrap Schottky diode. The bootstrap Schottky diode connected between the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) and 3.3V power supplies protects the ADSP-21160 DSPs from partially powering the 3.3V supply. Including a Schottky diode will shorten the delay between the supply ramps and thus prevent damage to the ESD diode protection circuitry. With this technique, of the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) rail rises ahead of the 3.3V rail, the Schottky diode pulls the 3.3V rail along with the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) rail.

Figure 11-5 shows a basic block diagram of the Schottky diode connected between the core and I/O voltage regulators and the DSP.

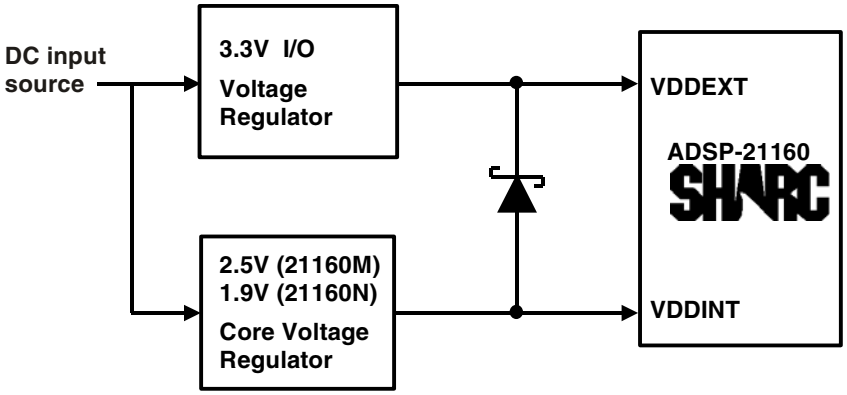


Figure 11-5. Dual +2.5V DC (21160M) or +1.9V DC (21160N)/3/3V Supplies with Schottky Diode

The anode of the diode must be connected to the +2.5V DC (ADSP-21160M DSP) or +1.9V DC (ADSP-21160N DSP) supply. The diode must have a forward biased voltage of 0.6V or less and must have a current rating sufficient to supply the 3.3V rail of the system. The use of a Schottky diode is the recommended method suggested by Analog Devices.

For recommendations on managing power-up sequencing for the core I/O dual voltage supply, refer to the “Power-up Sequencing” specifications in the *ADSP-21535 SHARC DSP Microcomputer Data Sheet*.

Designing for JTAG Emulation

The DSP Analog Devices DSP Tools product line of JTAG emulator is a development tool for debugging programs running in real time on DSP target system hardware. The Analog Devices DSP Tools product line of JTAG emulators provides a controlled environment for observing, debugging, and testing activities in a target system by connecting directly to the target processor through its JTAG interface.

Because the Analog Devices DSP Tools product line of JTAG emulator controls the target system’s DSP through the processor’s IEEE 1149.1 JTAG Test Access Port (TAP), non-intrusive in-circuit emulation is assured. The emulator uses the TAP to access the internals of the DSP, allowing the developer to load code, set breakpoints, observe variables, observe memory, examine registers, etc. The DSP must be halted to send data and commands, but once an operation is completed by the emulator, the DSP system is set running at full speed with no impact on system timings emulator does not impact target loading or timing. The emulator’s in-circuit probe connects to a variety of host computers (PCI bus, or USB) with plug-in boards.

Target systems must have a 14-pin connector in order to accept the Analog Devices DSP Tools product line of JTAG emulator in-circuit probe, a 14-pin plug.

Designs must add this connector to the target board if the board is intended for use with the ADSP-21535 JTAG Emulator. The total trace length between the JTAG connector and the furthest device sharing the Emulator’s JTAG pins should be limited to 15 inches maximum for guaranteed operation. This length restriction must include the emulator’s JTAG signals, which are routed to one or more ADSP-21535 devices, or a

combination of ADSP-21535 devices and other JTAG devices on the chain.

Target Board Connector

The emulator interface to an Analog Devices' JTAG DSP is a 14-pin header, as shown in Figure 11-6. The customer must supply this header on their target board in order to communicate with the emulator. The interface consists of a standard dual row 0.025" square post header, set on 0.1" x 0.1" spacing, with a minimum post length of 0.235". Pin 3 is the key position used to prevent the pod from being inserted backwards. This pin must be clipped on the target board.

The clearance (length, width, and height) around the header must be as shown in Figure 11-10. Maintain a minimum length of 0.15" and width of 0.10" for the target board header. The pod connector attaches the target board header in this area. Therefore, there must be clearance to attach and detach this connector. See the "DSP JTAG Pod Connector" on page 11-32 for detailed drawings of the pod connector.

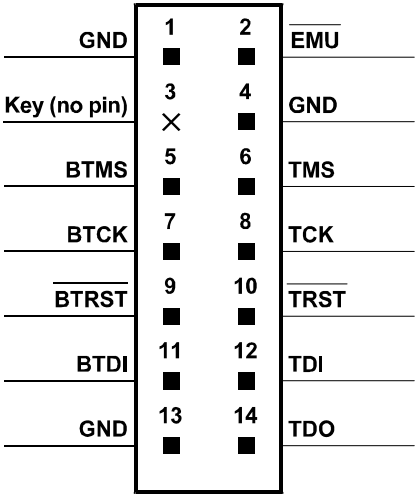


Figure 11-6. Emulator Interface for Analog Devices' JTAG DSPs

As can be seen in Figure 11-6, there are two sets of signals on the header, including the standard JTAG signals TMS, TCK, TDI, TDO, $\overline{\text{TRST}}$, $\overline{\text{EMU}}$ used for emulation purposes (via an emulator). Secondary JTAG signals BTMS, BTCK, BTDI, and $\overline{\text{BTRST}}$ are provided for optional use for board-level (boundary scan) testing. While they are rarely used, the "B" signals should be connected to a separate on-board JTAG boundary scan controller, if they are used. If the "B" signals will not be used, tie all of them to ground as shown in Figure 11-7.



BTCK can alternately be activated (for some older silicon) to VCC (+5V, +3.3V, or +2.5V) using a 4.7KΩ resistor, as described in previous documents. Tying the signal to ground is universal and will work for all silicon.

When the emulator is not connected to this header, jumpers should be placed across BTMS, BTCK, $\overline{\text{BTRST}}$, and BTDI as shown in Figure 11-7. This holds the JTAG signals in the correct state to allow the DSP to run freely.

All the jumpers should be removed when connecting the emulator to the JTAG header.

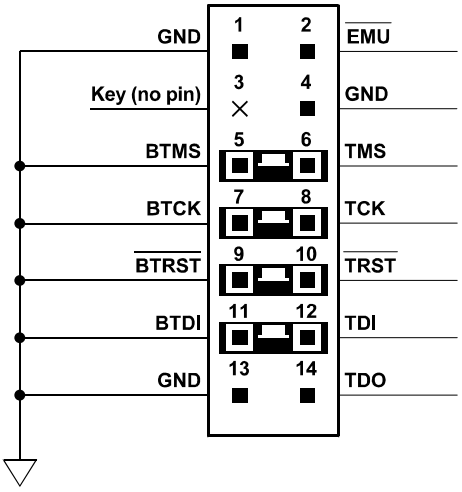


Figure 11-7. JTAG Target Board Connector With No Local Boundary Scan

For a list of the state of each standard JTAG signal refer to Table 11-5. Use the following legend: O = Output, I = Input, and NU = Not Used

Table 11-5. State of Standard JTAG Signals

Signal	Description	Emulator	DSP
TMS	Test Mode Select	O	I
TCK	Test Clock (10 MHz)	O	I
$\overline{\text{TRST}}$	Test Reset	O	I
TDI	Test Data In	O	I
TDO	Test Data Out	I	O
$\overline{\text{EMU}}$	Emulation Pin	I	O (Open Drain)
CLKIN	DSP Clock Input	NU	I

The DSP CLKIN signal is the clock signal line (typically 30 MHz or greater) that connects an oscillator to all DSPs in multiple DSP systems requiring synchronization. In order for synchronous DSP operations to work correctly the CLKIN signal on all the DSPs must be the same signal and the skew between them must be minimal (use clock drivers, or other means) - see the DSP users guide for more CLKIN details.

Note that the CLKIN signal is not used by the emulator and can cause noise problems if connected to the JTAG header. Legacy documents show it connected to pin 4 of the JTAG header. Pin 4 should be tied to ground on the 14-pin JTAG header (do not connect the JTAG header pin to the DSP CLKIN signal). If you have already connected it to the JTAG header pin, and are experiencing noise from this signal, simply clip this pin on the 14-pin JTAG header.

The final connections between a single DSP target and the emulation header (within 6 inches) are shown in Figure 11-8. A 4.7KΩ pull-up resistor has been added on TCK, TDI and TMS for increased noise resistance.

If a design uses more than one DSP (or other JTAG device in the scan chain), or if the JTAG header is more than 6 inches from the DSP, use a buffered connection scheme as shown in Figure 11-9 (no local boundary scan mode shown). To keep signal skew to a minimum, be sure the buffers are all in the same physical package (typical chips have 6, 8, or 16 drivers). Using a buffer that includes a series of resistors such as the 74ABT2244 family can reduce ringing on the JTAG signal lines.

For low voltage applications (3.3V, 2.5V, and 1.9V I/O), the 74ALVT, and 74AVC logic families is useful. Also, note the position of the pull-up resistor on $\overline{\text{EMU}}$. This is required since the $\overline{\text{EMU}}$ line is an open drain signal.

If more than one DSP (or JTAG device) is on the target (in the scan chain), you must buffer the JTAG header. This will keep the signals clean and avoid noise problems that occur with longer signal traces (ultimately resulting in reliable emulator operation).

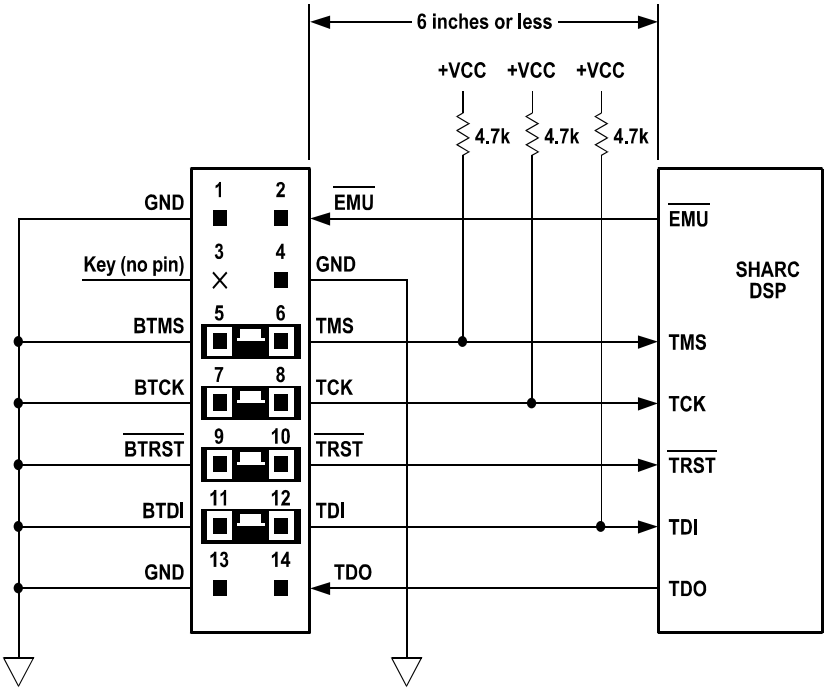


Figure 11-8. Single DSP Connection to the JTAG Header

Although the theoretical number of devices that can be supported (by the software) in one JTAG scan chain is large (50 devices or more) it is not recommended that you use more than eight physical devices in one scan chain. A physical device could however contain many JTAG devices such as inside a multi-chip module. The recommendation of not more than eight physical devices is mostly due to the transmission line effects that appear in long signal traces, and based on some field-collected empirical data.

The best approach for large numbers of physical devices is to break the chain into several smaller independent chains, each with their own JTAG header and buffer. If this is not possible, at least add some jumpers that

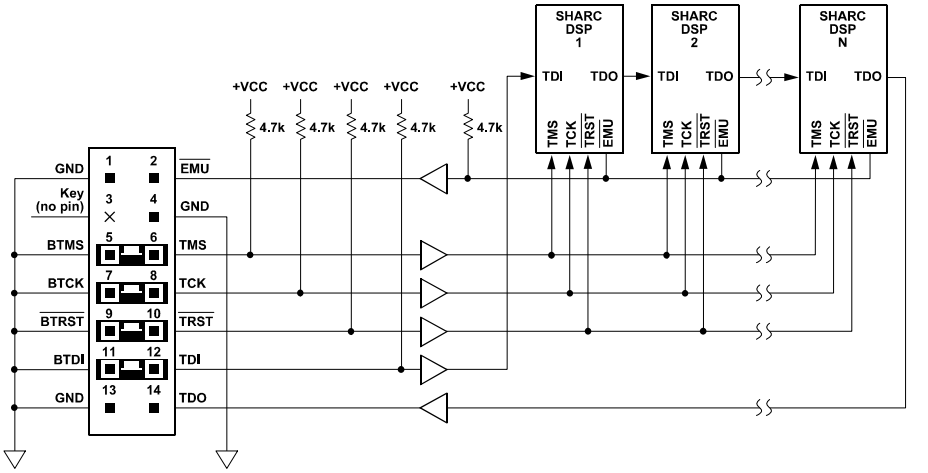


Figure 11-9. Multiple DSP Connection to JTAG Header

can reduce the number of devices in one chain for debug purposes, and pay special attention in the layout stage for transmission line effects.

Layout Requirements

All JTAG signals (TCK, TMS, TDI, TDO, $\overline{\text{EMU}}$, $\overline{\text{TRST}}$) should be treated as critical route signals. Specify a controlled impedance requirement for each route (value depends on your circuit board, typically 50-75Ω). Keeping crosstalk and inductance to a minimum on these lines by using a good ground plane and by routing away from other high noise signals such as clock lines is also important. Keep these routes as short and clean as possible, and keep the bused signals (TMS, TCK, $\overline{\text{TRST}}$, $\overline{\text{EMU}}$) as close to the same length as possible.

The JTAG TAP relies on the state of the TMS line and the TCK clock signal. If these signals have glitches (due to ground bounce, crosstalk, etc.) unreliable emulator operation will result. When experiencing emulator problems, look at these signals using a

high-speed digital oscilloscope. These lines must be clean, and may require special termination schemes. If you are buffering the JTAG header (most customers will) you must provide signal termination appropriate for your target board (series, parallel, R/C, etc.).

Power Sequence for Emulation

The power-on sequence for your target and emulation system is as follows:

1. Apply power to the emulator first, then to the target board. This ensures that the JTAG signals are in the correct state for the DSP to run free.
2. Upon power-on, the emulator drives the $\overline{\text{TRST}}$ signal low, keeping the DSP TAP in the test-logic-reset state, until the emulation software takes control.

Removal of power should be done in reverse: Turn off power to the target board, then to the emulator.

Additional JTAG Emulator References

The IEEE 1149.1 JTAG standard is sponsored by the Test Technology Standards Committee of the IEEE Computer Society, and published by the IEEE. The latest versions at the time of this publication are IEEE Standard. 1149.1-1990 and IEEE Standard 1149.1a-1993. To order a copy, call the IEEE at 1 800 678 4333 in the US and Canada, 1 908 981 1393 outside of the US and Canada. Visit the IEEE standards web site at http://standards.ieee.org/.

Pod Specifications

This section contains design details on various emulator pod designs by the Analog Devices DSP Tools product line. The emulator pod is the device that connects directly to the DSP target board 14-pin JTAG header. See also *Engineer to Engineer Notes EE-68*.

DSP JTAG Pod Connector

Figure 11-10 details the dimensions of the JTAG pod connector at the 14-pin target end. Figure 11-11 displays the keep-out area for a target board header. The keep-out area allows the pod connector to properly seat onto the target board header. This board area should contain no components (chips, resistors, capacitors, etc.). The dimensions are referenced to the center of the 0.25" square post pin.

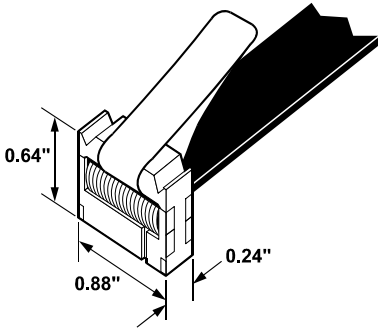


Figure 11-10. JTAG Pod Connector (14-pin Target End)

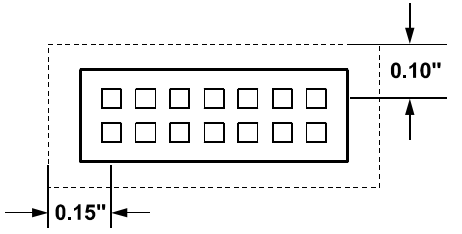


Figure 11-11. Keep-out Area For a Target Board Header

DSP 3.3V Pod Logic

A portion of the Analog Devices DSP Tools product line 3.3V emulator pod interface is shown in Figure 11-12. This figure describes the driver circuitry of the emulator pod. As can be seen, TMS, TCK and TDI are driven with a 33Ω series resistor. $\overline{\text{TRST}}$ is driven with a 100Ω series resistor. TDO and CLKIN are terminated with an optional 91/120Ω parallel terminator. $\overline{\text{EMU}}$ is pulled up with a 4.7KΩ resistor. The 74LVT244 chip drives the signals at 3.3V, with a maximum current rating of ±32mA.

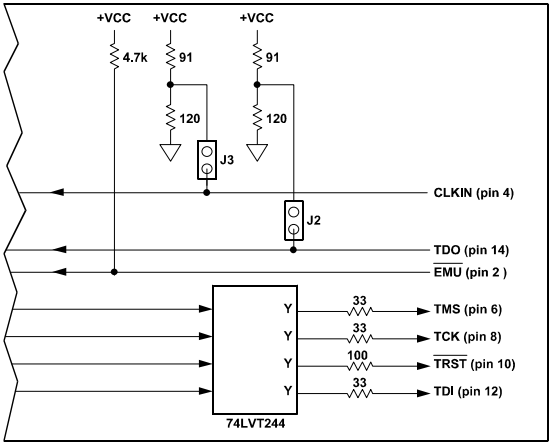


Figure 11-12. 3.3V JTAG Pod Driver Logic

Parallel terminate the TMS, TCK, $\overline{\text{TRST}}$, and TDI lines locally on your target board, if needed, since they are driven by the pod with sufficient current drive (±32mA). In order to use the terminators on the TDO line (CLKIN is not used), you MUST have a buffer on your target board JTAG header. The DSP is not capable of driving the parallel terminator load directly with TDO. Assuming the proper buffers are included, use the optional parallel terminators by placing a jumper on J2.

DSP 2.5V Pod Logic

A portion of the Analog Devices DSP Tools product line 2.5V emulator pod interface is shown in Figure 11-13. This figure describes the driver circuitry of the emulator pod. As can be seen, TMS, TCK, and TDI are driven with a 33Ω series resistor. $\overline{\text{TRST}}$ is driven with a 100Ω series resistor. TDO is pulled up with a 4.7KΩ resistor and terminated with an optional parallel terminator that can be configured by the user. $\overline{\text{EMU}}$ is pulled up with a 4.7KΩ resistor.

The CLKIN signal is not used and not connected inside the pod. The 74ALVT16244 chip drives the signals at 2.5V, with a maximum current rating of ±8mA.

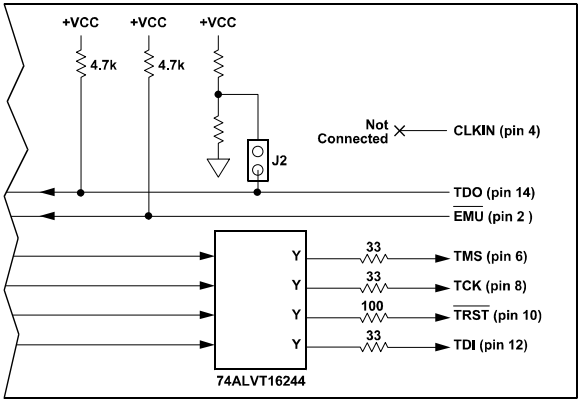


Figure 11-13. 2.5V JTAG Pod Driver Logic

You can terminate the TMS, TCK, $\overline{\text{TRST}}$, and TDI lines locally on your target board, if needed, as long as the terminator's current use does not exceed the driver's maximum current supply (±8mA). In order to use the terminator on the TDO line, include a buffer on your target board JTAG header. The DSP is not capable of driving a parallel terminator load (typically 50-75Ω) directly with TDO. Assuming you have the proper buffers, you may use the optional parallel terminator by adding the appropriate resistors and placing a jumper on J2.

Conditioning Input Signals

The DSP is a CMOS device. It has input conditioning circuits which simplify system design by filtering or latching input signals to reduce susceptibility to glitches or reflections.

The following sections describe why these circuits are needed and their effect on input signals.

A typical CMOS input consists of an inverter with specific N and P device sizes that cause a switching point of approximately 1.4V. This level is selected to be the midpoint of the standard TTL interface specification of V IL =0.8V and V IH =2.0V. This input inverter, unfortunately, has a fast response to input signals and external glitches wider than about 1 ns. Filter circuits and hysteresis are added after the input inverter on some DSP inputs, as described in the following sections.

Link Port Input Filter Circuits

The DSP's link port input signals have on-chip filter circuits rather than glitch rejection circuits. Filtering is not used on most signals because it delays the incoming signal and the timing specifications.

Filtering is implemented only on the link port data and clock inputs. This is possible because the link ports are self-synchronized. The clock and data are sent together. It is not the absolute delay but rather the relative delay between clock and data that determines performance margin.

By filtering both LxCLK and LxDAT3-0 with identical circuits, response to LxCLK glitches and reflections are reduced but relative delay is unaffected. The filter has the effect of ignoring a full strength pulse (a glitch) narrower than approximately 2 ns. Glitches that are not full strength can be somewhat wider. The link ports do not use glitch rejection circuits because they can be used with longer, series-terminated transmission lines where the reflections do not occur near the signal transitions.

RESET Input Hysteresis

Hysteresis is used only on the $\overline{\text{RESET}}$ input signal. Hysteresis causes the switching point of the input inverter to be slightly above 1.4V for a rising edge and slightly below 1.4V for a falling edge. The value of the hysteresis is approximately ± 0.1V. The hysteresis is intended to prevent multiple triggering of signals which are allowed to rise slowly, as might be expected on a reset line with a delay implemented by an RC input circuit. Hysteresis is not used to reduce the effect of ringing on DSP input signals with fast edges, because the amount of hysteresis that can be used on a CMOS chip is too small to make much difference. The small amount of hysteresis allowable is due to the restrictions on the tolerance of the V IL and V IH TTL input levels under worst case conditions. Refer to the DSP data sheet for exact specifications.

Designing For High Frequency Operation

Because the DSP processor can operate at very fast clock frequencies, signal integrity and noise problems must be considered for circuit board design and layout. The following sections discuss these topics and suggest various techniques to use when designing and debugging DSP systems.

Initial versions of the DSP are specified for operation at 50 MHz and 40 MHz internal clocks; the following information is based on these CLKIN frequencies. Refer to the most up-to-date DSP data sheet for current clock speed specifications.

All DSP synchronous behavior is specified to CLKIN. System designers are encouraged to clock synchronous peripherals/memory (which are attached to the DSP external port) with this same clock source (or a different low-skew output from the same clock driver). Alternatively, the clock output (CLKOUT) from the DSP may be employed to clock synchronous peripherals/memory.

Note the following behavior for CLKOUT:

- The DSP whose ID2-0=000 (uniprocessor), or 001 drives CLKOUT during reset.
- CLKOUT is specified relative to CLKIN in the DSP data sheet. When using this output to clock system components, the phase and jitter terms associated with this output must be treated as additional derating factors in determining specs.
- The bus master drives CLKOUT. In an MP system, this clock has multiple sources, including host logic, if present. The system component clocked by CLKOUT must be able to tolerate one or more cycles in which CLKOUT is held high, as bus ownership changes. Also, if the host logic operates asynchronously to the DSP CLKOUT frequency, the system component clocked by CLKOUT must be able to both operate at that host frequency, and handle the electrical characteristics of the CLKOUT transition to that asynchronous frequency domain.
- For systems not needing CLKOUT as a clock source, CLKOUT may be used to identify the current bus master. This requires that the outputs not be tied together. If and when this debug feature is not needed, the CLKOUT output can be disabled by setting the COD bit in the SYSCON register.

Clock Specifications and Jitter

The clock signal must be free of ringing and jitter. Clock jitter can easily be introduced in a system where more than one clock frequency exists.

High frequency jitter on the clock to the DSP may result in abbreviated internal cycles.

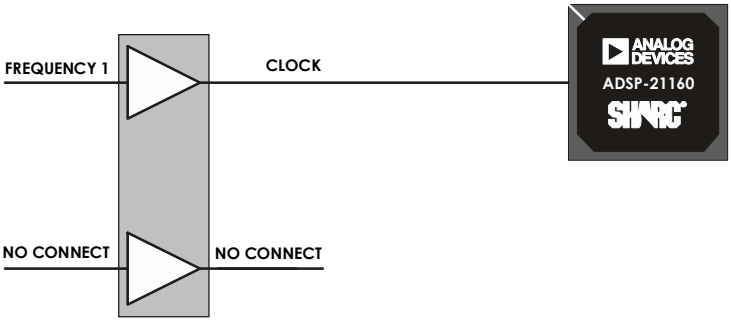


Figure 11-14. Reducing Clock Jitter and Ring

Never share a clock buffer IC with a signal of a different clock frequency. This introduces excessive jitter.

As shown in Figure 11-14, keep the portions of the system that operate at different frequencies as physically separate as possible.

Clock Distribution

There must be low clock skew between DSPs in a multiprocessor cluster when communicating synchronously on the external bus. The clock must be routed in a controlled-impedance transmission line that can be properly terminated at either the end of the line or the source.

Figure 11-15 illustrates end-of-line termination for the clock. End-of-line termination is not usually recommended unless the distance between the processors is extremely small, because devices that are at a different wire distance from each other receive a skewed clock. This is due to the propa-

gation delay of a PCB transmission line, which is typically 5 to 6 inches/ns.

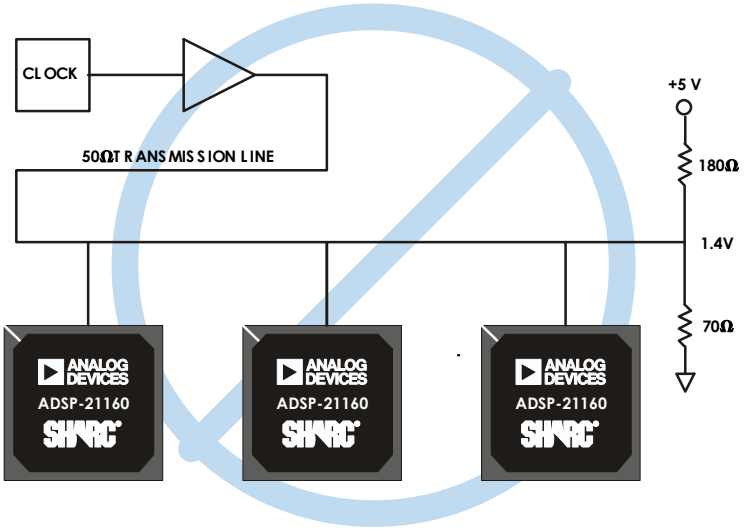


Figure 11-15. Do Not Use End-Of-Line Termination for the Clock

Figure 11-16 illustrates source termination for the clock. Source termination allows delays in each path to be identical. Each device must be at the end of the transmission line because only there does the signal have a single transition. The traces must be routed so that the delay through each is matched to the others. Line impedance higher than 50Ω may be used, but clock signal traces should be in the PCB layer closest to the ground plane to keep delays stable and crosstalk low. More than one device may be at the end of the line, but the wire length between them must be short and the impedance (capacitance) of these must be kept high. The matched inverters must be in the same IC and must be specified for a low skew (< 1

ns) with respect to each other. This skew should be as small as possible because it subtracts from the margin on most specifications.

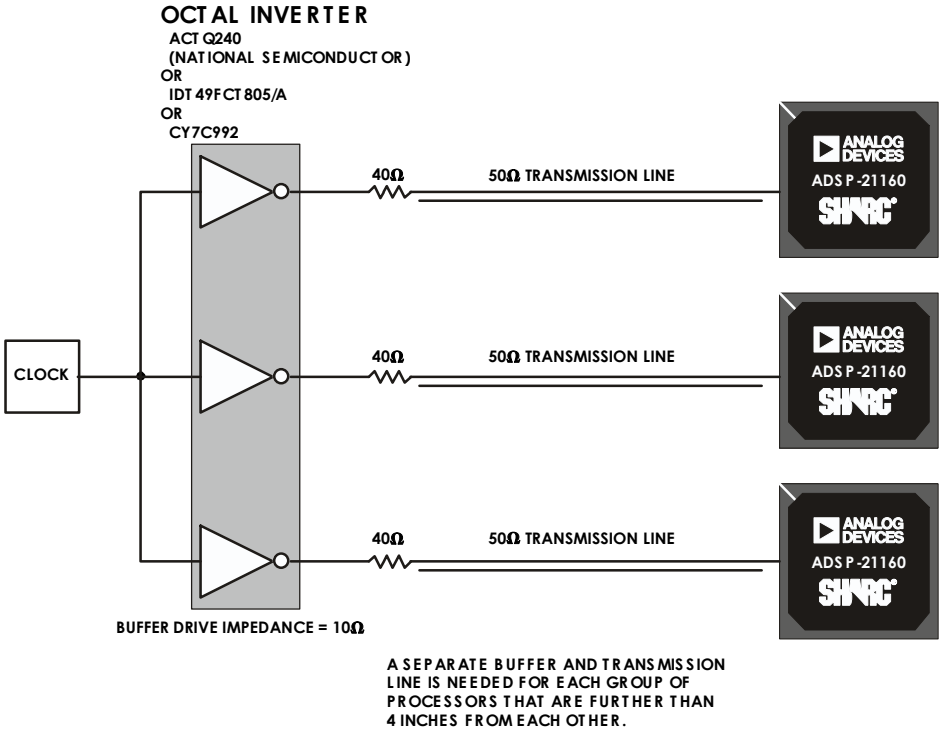


Figure 11-16. Use Source Termination to Distribute the Clock

Point-To-Point Connections

A series termination resistor may be added near the pin for point-to-point connections. This is typically used for link port applications when distances are greater than 6 inches as shown in Figure 11-17. For more specific guidance on related issues, see the reference source in "Recommended Reading" on page 11-47 for suggestions on transmission line termination and see the *ADSP-21160 SHARC DSP Microcomputer Data Sheet* for output drivers' rise and fall time data.

For link port operation at the full internal clock rate it is important to maintain low skew between the data (LxDAT7-0) and clock (LxCLK).

Although the DSP's serial ports may be operated at a slow rate, the output drivers still have fast edge rates and may require source termination for longer distances.

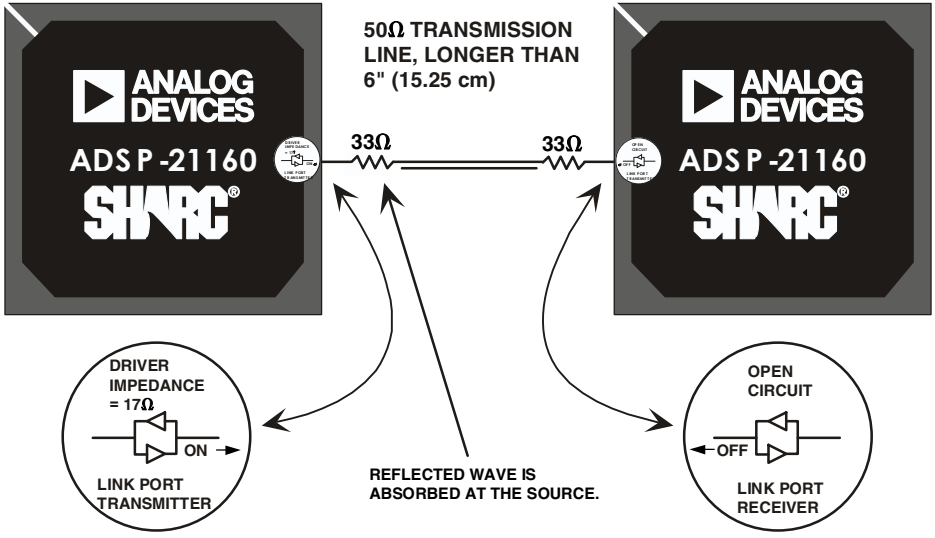


Figure 11-17. Source Termination For Point-To-Point Connections

Signal Integrity

The capacitive loading on high-speed signals should be reduced as much as possible. Loading of buses can be reduced by using a buffer for devices that operate with wait states, for example DRAMs. This reduces the capacitance on signals tied to the zero-wait-state devices, allowing these signals to switch faster and reducing noise-producing current spikes.

Signal run length (inductance) should also be minimized to reduce ringing. Extra care should be taken with certain signals such as the read and

write strobes (RDH/L, WRH/L) and acknowledge (ACK). In a multiprocessor cluster, each DSP can drive the read or write strobes. In this case, some damping resistance should be put in the signal path if the line length is greater than 6 inches. This is at the expense of additional signal delay. The time budget for these signals should be carefully analyzed.

Two possible damping arrangements between four DSPs are shown in Figure 11-18 and Figure 11-19.

In Figure 11-18, a star connection of resistors is used. Each DSP can drive the signal (for example, RDH/L or WRH/L strobe). Trace lengths should be minimized. Experiment with the optimal resistance value and placement; for example, near the processor or near the common connection. This adds signal delay.

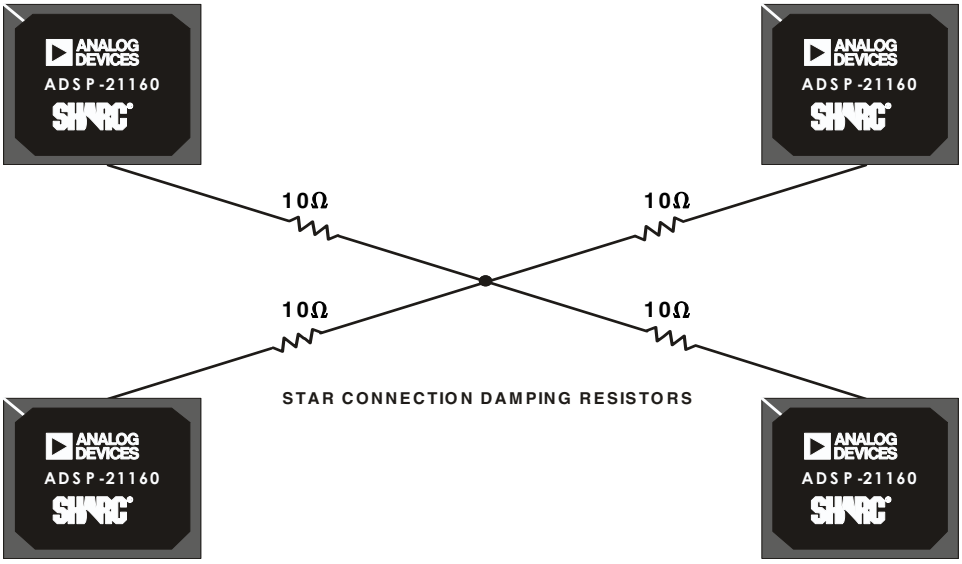


Figure 11-18. Star Connection Damping Resistors

In the example of Figure 11-19, where processors 1 and 2, and 3 and 4 are close to each other, a single damping resistor between the processor pairs

helps damp out reflections. Experiment with the resistor value. The two processor groups have a skew with respect to each other.

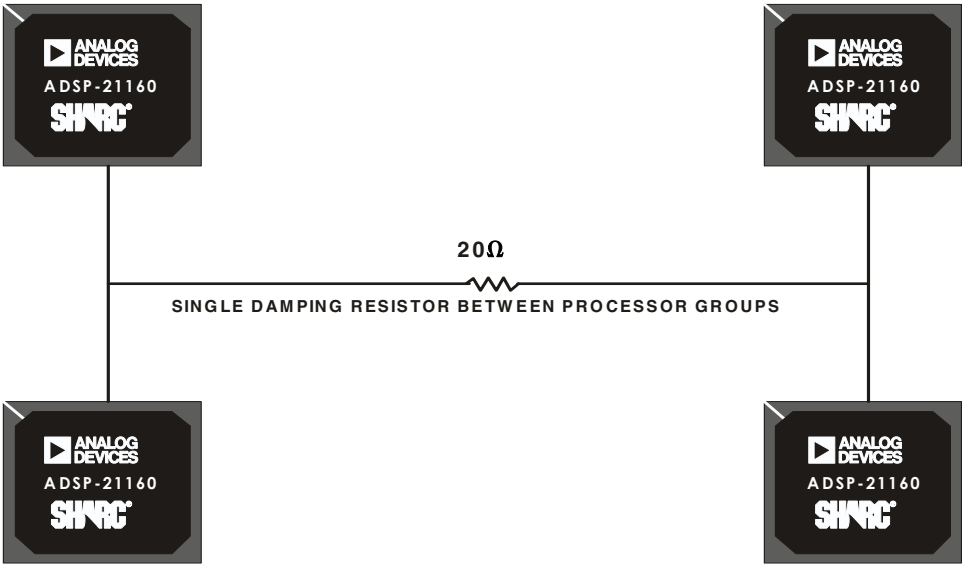


Figure 11-19. Single Damping Resistor Between Processor Groups

Another solution to multiple drivers where longer distances are involved is to have a single transmission line that is terminated at both ends. This arrangement is shown in Figure 11-20. The stubs to the processors must be kept as short as possible. Each device driver sees an impedance of 25Ω, but this resistor is biased at 1.4V, so the drive from the DSPs is sufficient for TTL levels. To reduce power dissipation in the system and in each DSP, this should only be used, if necessary, for signals such as the $\overline{\text{RDH/L}}$ or

$\overline{\text{WRH/L}}$ strobe. With this arrangement, the signals are skewed, but well behaved.

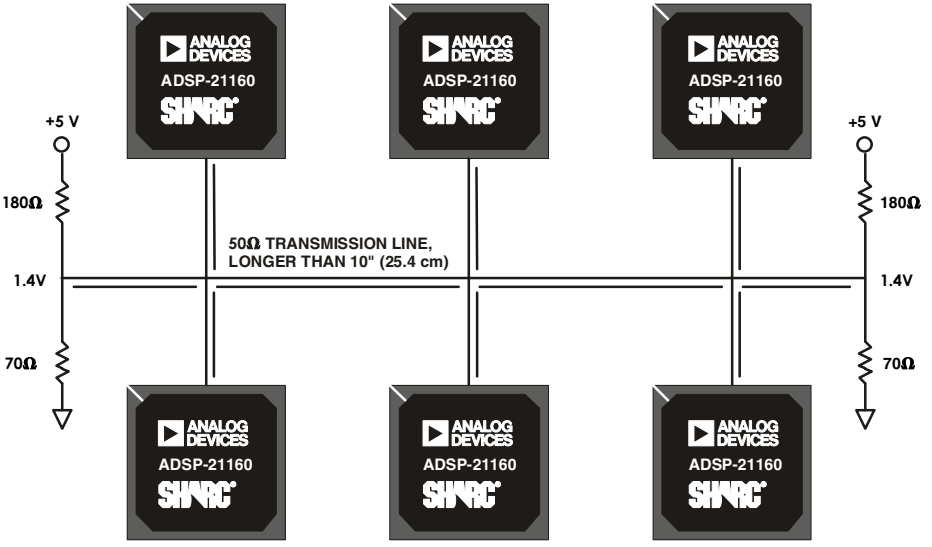


Figure 11-20. Single Transmission Line Terminated At Both Ends

Other Recommendations and Suggestions

These recommendations and suggestions are:

- Use more than one ground plane on the PCB to reduce crosstalk. Be sure to use lots of vias between the ground planes. One VDD plane for each supply is sufficient. These planes should be in the center of the PCB.
- Keep critical signals such as clocks, strobes, and bus requests on a signal layer next to a ground plane and away from or layout perpendicular to other non-critical signals to reduce crosstalk. For

example, data outputs switch at the same time that $\overline{\text{BRx}}$ inputs are sampled; if the layout permits crosstalk between them, the system could have problems with bus arbitration.

- Position the processors on both sides of the board to reduce area and distances if possible.
- Design for lower transmission line impedances to reduce crosstalk and to allow better control of impedance and delay.
- Use of 3.3V peripheral components and power supplies to help reduce transmission line problems, because the receiver switching voltage of 1.4V is close to the middle of the voltage swing. In addition, ground bounce and noise coupling is less.
- Experiment with the board and isolate crosstalk and noise issues from reflection issues. This can be done by driving a signal wire from a pulse generator and studying the reflections while other components and signals are passive.

Decoupling Capacitors and Ground Planes

Ground planes must be used for the ground and power supplies. Designs should use a minimum of eight bypass capacitors (six 0.1 µF and two 0.01 µF ceramic). The capacitors should be placed very close to the VDDEXT and VDDINT pins of the package as shown in Figure 11-21. Use short and fat traces for this. The ground end of the capacitors should be tied directly to the ground plane inside the package footprint of the DSP (underneath it, on the bottom of the board), not outside the footprint. A surface-mount capacitor is recommended because of its lower series inductance. Connect the power plane to the power supply pins directly with minimum trace length. The ground planes must not be densely perforated

with vias or traces as their effectiveness is reduced. In addition, there should be several large tantalum capacitors on the board.

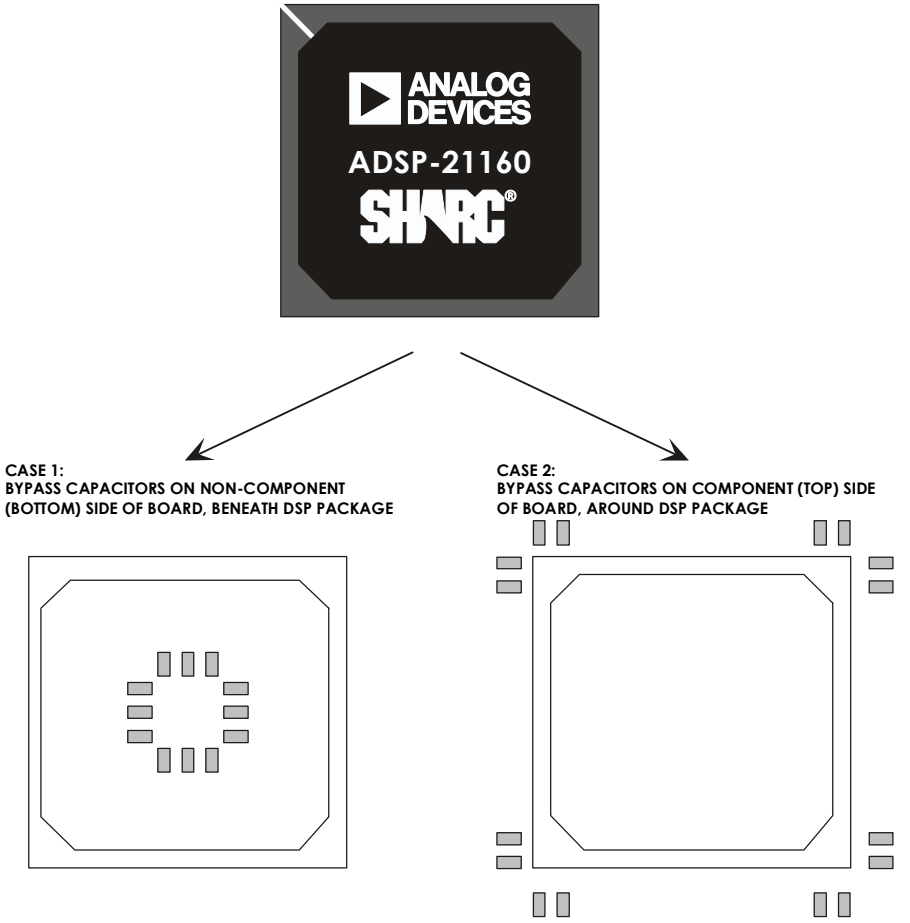


Figure 11-21. Bypass Capacitor Placement

Designs can use either bypass placement case shown in Figure 11-21 or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

Oscilloscope Probes

When making high-speed measurements, be sure to use a “bayonet” type or similarly short (< 0.5 inch) ground clip, attached to the tip of the oscilloscope probe. The probe should be a low-capacitance active probe with 3 pF or less of loading. The use of a standard ground clip with 4 inches of ground lead causes ringing to be seen on the displayed trace and makes the signal appear to have excessive overshoot and undershoot. A 1 GHz or better sampling oscilloscope is needed to see the signals accurately.

Recommended Reading

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high-frequency digital circuit design, and is an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines
- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables

- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson and Graham, Prentice Hall, Inc., ISBN 0-13-395724-1

Booting Single and Multiple Processors

Programs can be automatically downloaded to the internal memory of an DSP after power-up or after a software reset. This process is called booting. The DSP supports three booting modes: EPROM, host, and link port. For cases when the DSP must execute instructions from external memory without booting, a "No boot" mode may also be configured. For information on the setup and DMA processes for booting a single processor, see "Bootloading Through The External Port" on page 6-76 and "Bootloading Through The Link Port" on page 6-88.

Multiprocessor systems can be booted from a host processor, from external EPROM, through a link port, or from external memory.

Multiprocessor Host Booting

To boot multiple DSP processors from a host, each DSP must have its EBOOT, LBOOT, and $\overline{\text{BMS}}$ pins configured for host booting: EBOOT=0, LBOOT=0, and $\overline{\text{BMS}}$=1. After system powerup, each DSP is in the idle state

and the $\overline{\text{BRx}}$ bus request lines are deasserted. The host must assert the $\overline{\text{HBR}}$ input and boot each DSP by:

- Asserting its $\overline{\text{CS}}$ pin (for asynchronous)
- Writing to the multiprocessor memory space location (for synchronous)
- Downloading instructions as described in "Booting Another DSP" on page 7-116

Multiprocessor EPROM Booting

There are two methods of booting a multiprocessor system from an EPROM. Processors perform the following steps in these methods:

- Arbitrate for the bus
- DMA the 256 word boot stream, after becoming bus master
- Release the bus
- Execute the loaded instructions

All DSPs Boot in Turn from a Single EPROM

The $\overline{\text{BMS}}$ signals from each DSP may be wire-ORed together to drive the chip select pin of the EPROM. Each DSP can boot in turn, according to its priority. When the last one has finished booting, it must inform the others (which may be in the idle state) that program execution can begin (if all DSPs are to begin executing instructions simultaneously).

An example system that uses this processors-take-turns technique appears in Figure 11-22. When multiple DSPs boot from one EPROM, the DSPs can boot either identical code or different code from the EPROM. If the

processors load differing code, a jump table (based on processor ID) can be used to select the code for each processor.

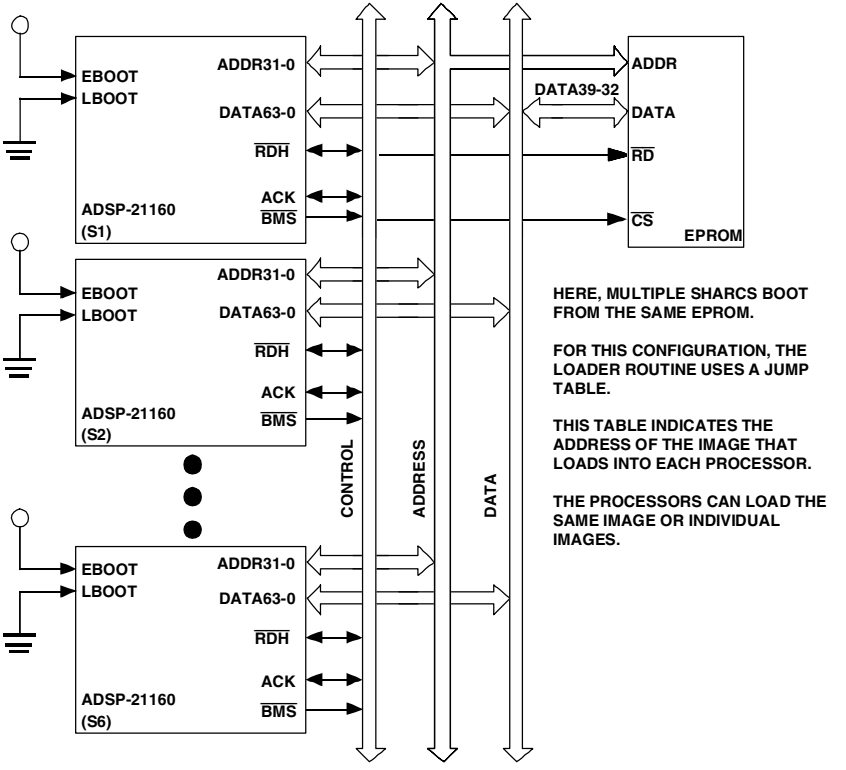


Figure 11-22. DSPs-Take-Turns Booting from an EPROM

One DSP is Booted, which then Boots the Others

The EBOOT pin of the DSP with IDx=1 must be set high for EPROM booting. All other DSPs should be configured for host booting (EBOOT=0, LBOOT=0, and $\overline{\text{BMS}}$=1), which leaves them in the idle state at startup and allows the DSP with IDx=1 to become bus master and boot itself. Only the $\overline{\text{BMS}}$ pin of DSP #1 is connected to the chip select of the EPROM. When

DSP #1 has finished booting, it can boot the remaining DSPs by writing to their external port DMA buffer 0 (EPB0) via multiprocessor memory space. An example system that uses this one-boots-others technique appears in Figure 11-23.

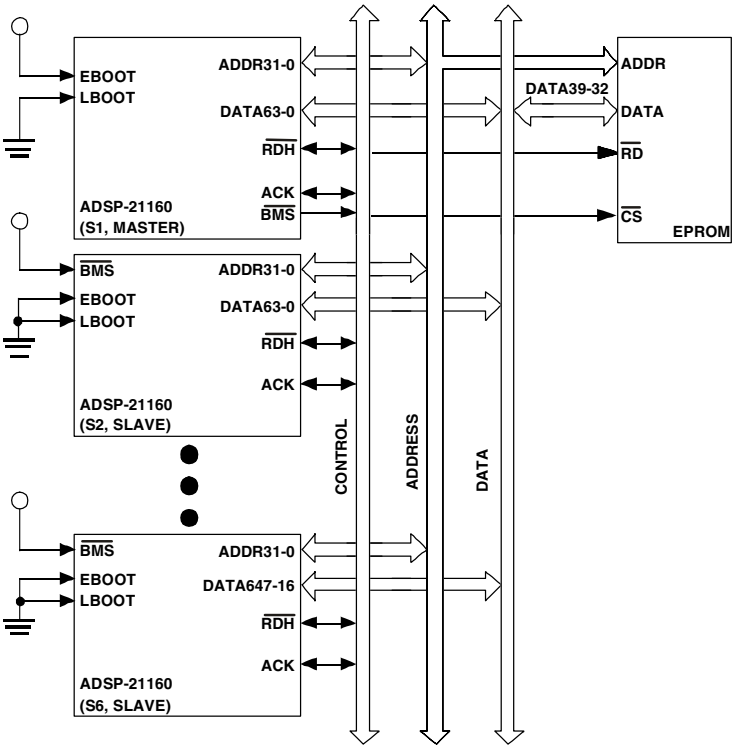


Figure 11-23. DSP-Boots-Others from an EPROM

Multiprocessor Link Port Booting

In systems where multiple DSPs are not connected by the parallel external bus, booting can be accomplished from a single source through the link ports. To simultaneously boot all of the DSPs, a parallel common connection should be made to link port buffer 4 (LBUF4) on each of the

processors. If only a daisy chain connection exists between the processors' link ports, then each DSP can boot the next one in turn. Link Buffer 4 must always be used for booting.

Multiprocessor Booting From External Memory

If external memory contains a program after reset, then the DSP with IDx=1 should be set up for no boot mode. It begins executing from address 0x0080 0004 in external memory. When booting has completed, the other DSPs may be booted by DSP #1 if they are set up for host booting, or they can begin executing out of external memory if they are set up for no boot mode. Multiprocessor bus arbitration allows this booting to occur in an orderly manner. The bus arbitration sequence after reset is described in "Multiprocessor Bus Arbitration" on page -101.

A REGISTERS

The DSP has general purpose and dedicated registers in each of its functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and (for I/O processor registers) memory mapped address. Information on each type of register is available at the following locations:

- "Control and Status System Registers" on page A-2
- "Processing Element Registers" on page A-15
- "Program Sequencer Registers" on page A-17
- "Data Address Generator Registers" on page A-31
- "I/O Processor Registers" on page A-33

When writing DSP programs, it is often necessary to set, clear, or test bits in the DSP's registers. While these bit operations can all be done by referring to the bit's location within a register or (for some operations) the register's address with a hexadecimal number, it is much easier to use symbols that correspond to the bit's or register's name. For convenience and consistency, Analog Devices provides a header file that provides these bit and registers definitions. For more information, see the "Register and Bit #Defines File (def21160.h)" on page A-82.



Many registers have reserved bits. When writing to a register, programs may only clear (write zero to) the register's reserved bits.

Control and Status System Registers

The DSP's control and status system registers configure how the processor core operates and indicate the status of many processor core operations. In the *ADSP-21160 SHARC DSP Instruction Set Reference*, these registers are referred to as System Registers (SREG), which are a subset of the DSP's Universal Registers (UREG). Not all registers are valid in all assembly language instructions. In the assembly syntax descriptions, the register group name (UREG, SREG, and others) indicates which type of register is valid within the instruction's context. Table A-1 lists the processor core's control and status registers with their initialization values. Descriptions of each register follow. Other system registers (SREG) are in the I/O processor. For more information, see "I/O Processor Registers" on page A-33.

Table A-1. Control and Status System Registers (SREG and UREG)

Register Name and Page Reference	Initialization After Reset
"Mode Control 1 Register (MODE1)" on page A-3	0x0000 0000
"Mode Mask Register (MMASK)" on page A-5	0x0020 0000
"Mode Control 2 Register (MODE2)" on page A-6	0xnn00 0000[1]
"Arithmetic Status Registers (ASTATx and ASTATy)" on page A-8	0x0000 0000
"Sticky Status Registers (STKYx and STKYy)" on page A-8	0x0540 0000
"User-Defined Status Registers (USTATx)" on page A-15	0x0000 0000

1 MODE2 bits 31-25 are the processor ID and silicon revision number, so the initialization value varies with the DSP's ID2-0 pins' input and the silicon revision.

Mode Control 1 Register (MODE1)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for this register is 0x0000 0000.

Table A-2. Mode Control 1 Register (MODE1) Bit Definitions

Bit(s)	Name	Definition
0	BR8	**Bit Reverse Addressing For Index I8 Enable.** This bit enables (bit reversed if set, =1) or disables (normal if cleared, =0) bit reversed addressing for accesses that are indexed with DAG2 register I8.
1	BR0	**Bit Reverse Addressing For Index I0 Enable.** This bit enables (bit reversed if set, =1) or disables (normal if cleared, =0) bit reversed addressing for accesses that are indexed with DAG1 register I0.
2	SRCU	**Secondary Registers For Computational Units Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary result (MR) registers in the computational units.
3	SRD1H	**Secondary Registers For DAG1 High Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary DAG1 registers for the upper half (I, M, L, B7-4) of the address generator.
4	SRD1L	**Secondary Registers For DAG1 Low Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary DAG1 registers for the lower half (I, M, L, B3-0) of the address generator.
5	SRD2H	**Secondary Registers For DAG2 High Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary DAG2 registers for the upper half (I, M, L, B15-12) of the address generator.
6	SRD2L	**Secondary Registers For DAG2 Low Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary DAG2 registers for the lower half (I, M, L, B11-8) of the address generator.
7	SRRFH	**Secondary Registers For Register File High Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary data registers for the upper half (R15-8) of the computational units.

Table A-2. Mode Control 1 Register (MODE1) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
9-8		Reserved
10	SRRFL	**Secondary Registers For Register File Low Enable.** This bit enables (use secondary if set, =1) or disables (use primary if cleared, =0) secondary data registers for the lower half (R7-0) of the computational units.
11	NESTM	**Nesting Multiple Interrupts Enable.** This bit enables (nest if set, =1) or disables (no nesting if cleared, =0) interrupt nesting in the interrupt controller. When interrupt nesting is disabled, a higher priority interrupt can not interrupt a lower priority interrupt's service routine. Other interrupts are latched as they occur, but the DSP process them after the active routine finishes. When interrupt nesting is enabled, a higher priority interrupt can interrupt a lower priority interrupt's service routine. Lower interrupts are latched as they occur, but the DSP process them after the nested routines finish.
12	IRPTEN	**Global Interrupt Enable.** This bit enables (if set, =1) or disables (if cleared, =0) all maskable interrupts.
13	ALUSAT	**ALU Saturation Select.** This bit selects whether the computational units saturate results on positive or negative fixed-point overflows (if 1) or return unsaturated results (if 0).
14	SSE	**Fixed-point Sign Extension Select.** This bit selects whether the computational units sign extend short-word, 16-bit data (if 1) or zero-fill the upper 32 bits (if 0).
15	TRUNC	**Truncation Rounding Mode Select.** This bit selects whether the computational units round results with round-to-zero (if 1) or round-to-nearest (if 0).
16	RND32	**Rounding For 32-bit Floating-point Data Select.** This bit selects whether the computational units round floating-point data to 32 bits (if 1) or round to 40 bits (if 0).
18-17	CSEL	**Bus Master Code Selection.** These bits indicate whether the DSP processor has control of the external bus as follows: 00=DSP is bus master or 01, 10, 11=DSP is not bus master.
20-19		Reserved

Table A-2. Mode Control 1 Register (MODE1) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
21	PEYEN	**Processor Element Y Enable.** This bit enables computations in PEy—SIMD mode—(if 1) or disables PEy—SISD mode—(if 0). When set, Processing Element Y (computation units and register files) accepts instruction dispatches. When cleared, Processing Element Y goes into a low power mode.
22	BDCST9	**Broadcast Register Loads Indexed With I9 Enable.** This bit enables (broadcast I9 if set, =1) or disables (no I9 broadcast if cleared, =0) broadcast register loads for loads that use the data address generator I9 index. When the BDCST9 bit is set, data register loads from the PM data bus that use the I9 DAG2 index register are "broadcast" to a register or register pair in each PE.
23	BDCST1	**Broadcast Register Loads Indexed With I1 Enable.** This bit enables (broadcast I1 if set, =1) or disables (no I1 broadcast if cleared, =0) broadcast register loads for loads that use the data address generator I1 index. When the BDCST1 bit is set, data register loads from the DM data bus that use the I1 DAG1 index register are "broadcast" to a register or register pair in each PE.
24	CBUFEN	**Circular Buffer Addressing Enable.** This bit enables (circular if set, =1) or disables (linear if cleared, =0) circular buffer addressing for buffers with loaded I, M, B, and L data address generator register.
31-25		Reserved

Mode Mask Register (MMASK)

This is a non-memory mapped, universal, system register (`UREG` and `SREG`). The reset value for this register is `0x0020 0000`.

Each bit in the `MMASK` register corresponds to a bit in the `MODE1` register. Bits that are set in `MMASK` are used to clear bits in `MODE1` when the DSP's status stack is pushed. This effectively disables different modes upon servicing an interrupt, or when executing a Push Sts instruction.

The DSP's status stack will be pushed in two cases:

- When you execute a Push Sts instruction explicitly in your code.
- When an $\overline{\text{IRQ}}$2-0 timer expires or a VIRPT interrupt occurs.

Example:

Before the Push Sts instruction, MODE1 is set to 0x01202811. This MODE1 value corresponds to the following settings being enabled:

- Bit Reversing for I8
- Secondary Registers for DAG2 (high)
- Interrupt Nesting, ALU Saturation
- Processor Element Y (SIMD)
- Circular Buffering

MMASK is set to 0x0020 2001 indicating that you want to disable ALU Saturation, SIMD, and bit reversing for I8 after pushing the status stack. The value in MODE1 after Push Sts is 0x0100 0810. The other settings that were previously in MODE1 remain the same. The only bits that are affected are those that are set both in MMASK and in MODE1. These bits are cleared after the status stack is pushed.

Note also that the reset value of MMASK is 0x0020 0000. If you do not make any changes to the MMASK register, the default setting will automatically disable SIMD when servicing any of the hardware interrupts mentioned above, or during any push of the status stack.

Mode Control 2 Register (MODE2)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for this register is 0xmm00 0000. Because bits 31-25 in this register are the DSP ID and silicon revision, the reset value varies with the

system setting and silicon revision. Bits 31-25 of the MODE2 register are also readable in the MODE2_SHDW register. For more information, see "MODE2 Shadow Register (MODE2_SHDW)" on page A-53.

Table A-3. Mode Control 2 Register (MODE2) Bit Definitions

Bit	Name	Definition
0	IRQ0E	$\overline{\text{IRQ0}}$ Sensitivity Select. This bit selects sensitivity for $\overline{\text{IRQ0}}$ as edge-sensitive (if set, =1) or level-sensitive (if cleared, =0).
1	IRQ1E	$\overline{\text{IRQ1}}$ Sensitivity Select. This bit selects sensitivity for $\overline{\text{IRQ1}}$ as edge-sensitive (if set, =1) or level-sensitive (if cleared, =0).
2	IRQ2E	$\overline{\text{IRQ2}}$ Sensitivity Select. This bit selects sensitivity for $\overline{\text{IRQ2}}$ as edge-sensitive (if set, =1) or level-sensitive (if cleared, =0).
3		Reserved
4	CADIS	**Cache Disable.** This bit disables the instruction cache (if set, =1) or enables the cache (if cleared, =0).
5	TIMEN	**Timer Enable.** This bit enables the timer (starts, if set, =1) or disables the timer (stops, if cleared, =0).
6	BUSLK	**Bus Lock Request.** This bit requests bus lock (DSP retains bus mastership, if set, =1) or does not request bus lock (normal bus mastering, if cleared, =0).
14-7		Reserved
15	FLG0O	**FLAG0 Output Select.** This bit selects the I/O direction for FLAG0 as an output (if set, =1) or an input (if cleared, =0).
16	FLG1O	**FLAG1 Output Select.** This bit selects the I/O direction for FLAG1 as an output (if set, =1) or an input (if cleared, =0).
17	FLG2O	**FLAG2 Output Select.** This bit selects the I/O direction for FLAG2 as an output (if set, =1) or an input (if cleared, =0).
18	FLG3O	**FLAG3 Output Select.** This bit selects the I/O direction for FLAG3 as an output (if set, =1) or an input (if cleared, =0).
19	CAFRZ	**Cache Freeze.** This bit freezes the instruction cache (retains contents, if set, =1) or thaws the cache (allows new input, if cleared, =0).

Table A-3. Mode Control 2 Register (MODE2) Bit Definitions (Cont'd)

Bit	Name	Definition
20	IRAE	**I/O Processor Register Access Enable.** This bit enables detection of I/O processor register accesses (if set, =1) or disables detection of I/O processor register accesses (if cleared, =0). If IRAE is set, the DSP flags an access by setting the IRA bit in the STKYx register. For more information, see "IRA" on page A-14.
21	U64MAE	**Unaligned 64-bit Memory Access Enable.** This bit enables detection of unaligned long word accesses (if set, =1) or disables detection of unaligned long word accesses (if cleared, =0). If U64MAE is set, the DSP flags an unaligned long word accesses by setting the U64MA bit in the STKYx register. For more information, see "U64MA" on page A-14.
31-22		Reserved

Arithmetic Status Registers (ASTATx and ASTATy)

These are non-memory mapped, universal, system registers (UREG and SREG). The reset value for these registers is 0x0000 0000. Each processing element has its own ASTAT register. ASTATx indicates status for PEx operations, and ASTATy indicates status for PEy operations (see Table A-4).



If a program loads the ASTATx register manually, there is a one cycle effect latency before the new value in ASTATx can be used in a conditional instruction

Sticky Status Registers (STKYx and STKYy)

These are non-memory mapped, universal, system registers (UREG and SREG). The reset value for these registers is 0x0000 0000. Each processing element has its own STKY register. STKYx indicates status for PEx operations and indicates status for some program sequencer stacks. The STKYy register

Table A-4. Arithmetic Status Registers (ASTATx/y) Bit Definitions

Bit(s)	Name	Definition
0	AZ	**ALU Zero/Floating-Point Underflow.** This bit indicates whether the last ALU operation's result was zero (if set, =1) or non-zero (if cleared, =0). The ALU updates AZ for all fixed-point and floating-point ALU operations. AZ can also indicate a floating-point underflow. During an ALU underflow—indicated by a set (=1) AUS bit in the STKYx/y register, the DSP sets AZ if the floating-point result is smaller than can be represented in the output format.
1	AV	**ALU Overflow.** This bit indicates whether the last ALU operation's result overflowed (if set, =1) or did not overflow (if cleared, =0). The ALU updates AV for all fixed-point and floating-point ALU operations. For fixed-point results, the DSP sets AV and the AOS bit in the STKYx/y register when the XOR of the two most significant bits is a 1. For floating-point results, the DSP sets AV and the AVS bit is the STKYx/y register when the rounded result overflows (unbiased exponent > 127).
2	AN	**ALU Negative.** This bit indicates whether the last ALU operation's result was negative (if set, =1) or positive (if cleared, =0). The ALU updates AN for all fixed-point and floating-point ALU operations.
3	AC	**ALU fixed-point Carry.** This bit indicates whether the last ALU operation had a carry out of most significant bit of the result (if set, =1) or had no carry (if cleared, =0). The ALU updates AC for all fixed-point operations. The DSP clears AC during fixed-point logic operations: PASS, MIN, MAX, COMP, ABS, and CLIP. The ALU reads the AC flag for fixed-point accumulate operations: addition with carry and fixed-point subtraction with carry.
4	AS	**ALU X-Input Sign (for ABS and MANT).** This bit indicates whether the last ALU ABS or MANT operation's input was negative (if set, =1) or positive (if cleared, =0). The ALU updates AS for only fixed-point and floating-point ABS and the MANT operations. The ALU clears AS for all operations other than ABS and MANT.

Table A-4. Arithmetic Status Registers (ASTATx/y)
Bit Definitions (Cont'd)

Bit(s)	Name	Definition
5	AI	**ALU Floating-Point Invalid Operation.** This bit indicates whether the last ALU operation's input was invalid (if set, =1) or valid (if cleared, =0). The ALU updates AI for all fixed-point and floating-point ALU operations. The DSP sets AI and the AIS bit in the STKYx/y register if the ALU operation: • Receives a NAN input operand • Adds opposite-signed Infinities • Subtracts like-signed Infinities • Overflows during a floating-point to fixed-point conversion when saturation mode is not set • Operates on an Infinity when saturation mode is not set
6	MN	**Multiplier Negative.** This bit indicates whether the last multiplier operation's result was negative (if set, =1) or positive (if cleared, =0). The multiplier updates MN for all fixed-point and floating-point multiplier operations.
7	MV	**Multiplier Overflow.** This bit indicates whether the last multiplier operation's result overflowed (if set, =1) or did not overflow (if cleared, =0). The multiplier updates MV for all fixed-point and floating-point multiplier operations. For floating-point results, the DSP sets MV and the MVS bit in the STKYx/y register if the rounded result overflows (unbiased exponent > 127). For fixed-point results, the DSP sets MV and the MOS bit in the STKYx/y register if the result of the multiplier operation is: • Twos-complement, fractional with the upper 17 bits of MR not all zeros or all ones • Twos-complement, integer with the upper 49 bits of MR not all zeros or all ones • Unsigned, fractional with the upper 16 bits of MR not all zeros • Unsigned, integer with the upper 48 bits of MR not all zeros If the multiplier operation directs a fixed-point result to an MR register, the DSP places the overflowed portion of the result in MR1 and MR2 for an integer result or places it in MR2 only for a fractional result.

Table A-4. Arithmetic Status Registers (ASTATx/y)
Bit Definitions (Cont'd)

Bit(s)	Name	Definition
8	MU	**Multiplier Floating-Point Underflow.** This bit indicates whether the last multiplier operation's result underflowed (if set, =1) or did not underflow (if cleared, =0). The multiplier updates MU for all fixed-point and floating-point multiplier operations. For floating-point results, the DSP sets MU and the MUS bit in the STKYx/y register if the floating-point result underflows (unbiased exponent < –126). Denormal operands are treated as Zeros, therefore they never cause underflows. For fixed-point results, the DSP sets MU and the MUS bit in the STKYx/y register if the result of the multiplier operation is: • Twos-complement, fractional: upper 48 bits all zeros or all ones, lower 32 bits not all zeros • Unsigned, fractional: upper 48 bits all zeros, lower 32 bits not all zeros If the multiplier operation directs a fixed-point, fractional result to an MR register, the DSP places the underflowed portion of the result in MR0.
9	MI	**Multiplier Floating-Point Invalid Operation.** This bit indicates whether the last multiplier operation's input was invalid (if set, =1) or valid (if cleared, =0). The multiplier updates MI for floating-point multiplier operations. The DSP sets MI and the MIS bit in the STKYx/y register if the ALU operation: • Receives a NAN input operand • Receives an Infinity and Zero as input operands
10	AF	**ALU Floating-Point Operation.** This bit indicates whether the last ALU operation was floating-point (if set, =1) or fixed-point (if cleared, =0). The ALU updates AF for all fixed-point and floating-point ALU operations.
11	SV	**Shifter Overflow.** This bit indicates whether the last shifter operation's result overflowed (if set, =1) or did not overflow (if cleared, =0). The shifter updates SV for all shifter operations. The DSP sets SV if the shifter operation: • Shifts the significant bits to the left of the 32-bit fixed-point field • Tests, sets, or clears a bit outside of the 32-bit fixed-point field • Extracts a field that is past or crosses the left edge of the 32-bit fixed-point field • Performs a LEFTZ or LEFTO operation that returns a result of 32

Table A-4. Arithmetic Status Registers (ASTATx/y)
Bit Definitions (Cont'd)

Bit(s)	Name	Definition
12	SZ	**Shifter Zero.** This bit indicates whether the last shifter operation's result was zero (if set, =1) or non-zero (if cleared, =0). The shifter updates SZ for all shifter operations. The DSP also sets SZ if the shifter operation performs a bit test on a bit outside of the 32-bit fixed-point field.
13	SS	**Shifter Input Sign.** This bit indicates whether the last shifter operation's input was negative (if set, =1) or positive (if cleared, =0). The shifter updates SS for all shifter operations.
17-14		Reserved
18	BTF	**Bit Test Flag for System Registers.** This bit indicates whether the last system register bit manipulation operation Bit Tst operation was true (if set, =1) or false (if cleared, =0). The DSP sets BTF when the bit(s) in a system register and value in the Bit Tst instruction match. The DSP also sets BTF when the bit(s) in a system register and value in the Bit Xor instruction match.
23-19		Reserved
31-24	CACC	**Compare Accumulation Shift Register.** Bit 31 of CACC indicates which operand was greater during the last ALU compare operation: X input (if set, =1) or Y input (if cleared, =0). The other seven bits in CACC form a right-shift register, each storing a previous compare accumulation result. With each new compare, the DSP right shifts the values of CACC, storing the newest value in bit 31 and the oldest value in bit 24.

only indicates status for PEy operations.Table A-5 lists bits for both STKYx and STKYy, noting with an 7 the bits that apply only to STKYx.



STKY bits do not clear themselves after the condition they flag is no longer true. They remain "sticky" until cleared by the program.

The DSP sets a STKY bit in response to a condition. For example, the DSP sets the AUS bit in the STKY register when an ALU underflow set AZ in the ASTAT register. The DSP clears AZ if the next ALU operation does not cause an underflow, but AUS remains set until a program clears the STKY bit. Interrupt service routines should clear their interrupt's corresponding

STKY bit so the DSP can detect a re-occurrence of the condition. For example, an interrupt service routine for the floating-point underflow exception interrupt (FLTUI) would clear the AUS bit in the STKY register near the beginning of the routine.

Table A-5. Sticky Status Registers (STKYx/y) Bit Definitions

Bit(s)	Name	Definition	At right: + shows bits in both STKYx/y x shows bits in STKYx only ↓
0	AUS	**ALU Floating-Point Underflow.** This bit is a sticky indicator for the ALU AS bit. For more information, see "AZ" on page A-9.	+
1	AVS	**ALU Floating-Point Overflow.** This bit is a sticky indicator for the ALU AV bit. For more information, see "AV" on page A-9.	+
2	AOS	**ALU Fixed-Point Overflow.** This bit is a sticky indicator for the ALU AV bit. For more information, see "AV" on page A-9.	+
4-3		Reserved	
5	AIS	**ALU Floating-Point Invalid Operation.** This bit is a sticky indicator for the ALU AI bit. For more information, see "AI" on page A-10.	+
6	MOS	**Multiplier Fixed-Point Overflow.** This bit is a sticky indicator for the multiplier MV bit. For more information, see "MV" on page -10.	+
7	MVS	**Multiplier Floating-Point Overflow.** This bit is a sticky indicator for the multiplier MV bit. For more information, see "MV" on page -10.	+
8	MUS	**Multiplier Floating-Point Underflow.** This bit is a sticky indicator for the multiplier MU bit. For more information, see "MU" on page -11.	+
9	MIS	**Multiplier Floating-Point Invalid Operation.** This bit is a sticky indicator for the multiplier MI bit. For more information, see "MI" on page -11.	+
16-10		Reserved	

Table A-5. Sticky Status Registers (STKYx/y) Bit Definitions (Cont'd)

Bit(s)	Name	Definition	At right: + shows bits in both STKYx/y ↓ x shows bits in STKYx only ↓
17	CB7S	**DAG1 Circular Buffer 7 Overflow.** This bit indicates whether a circular buffer being addressed with DAG1 register I7 has overflowed (if set, =1) or has not overflowed (if cleared, =0). A circular buffer overflow occurs when DAG circular buffering operation increments the I register past the end of buffer.	x
18	CB15S	**DAG2 Circular Buffer 15 Overflow.** This bit indicates whether a circular buffer being addressed with DAG2 register I15 has overflowed (if set, =1) or has not overflowed (if cleared, =0). A circular buffer overflow occurs when DAG circular buffering operation increments the I register past the end of buffer.	x
19	IRA	**IOP Register Access.** This bit indicates whether a core, host, or multiprocessor access to I/O processor registers has occurred (if 1) or has not occurred (if 0).	x
20	U64MA	**Unaligned 64-bit Memory Access.** This bit indicates whether a Normal word access with the LW mnemonic addressing an uneven memory address has occurred (if 1) or has not occurred (if 0).	x
21	PCFL	**PC Stack Full.** This bit indicates whether the PC stack is full (if 1) or not full (if 0)—Not a sticky bit, cleared by a Pop.	x
22	PCEM	**PC Stack Empty.** This bit indicates whether the PC stack is empty (if 1) or not empty (if 0)—Not sticky, cleared by a Push.	x
23	SSOV	**Status Stack Overflow.** This bit indicates whether the status stack is overflowed (if 1) or not overflowed (if 0)—A sticky bit.	x
24	SSEM	**Status Stack Empty.** This bit indicates whether the status stack is empty (if 1) or not empty (if 0)—Not sticky, cleared by a Push.	x
25	LSOV	**Loop Stack Overflow.** This bit indicates whether the loop counter stack and loop stack are overflowed (if 1) or not overflowed (if 0)—A sticky bit.	x
26	LSEM	**Loop Stack Empty.** This bit indicates whether the loop counter stack and loop stack are empty (if 1) or not empty (if 0)—Not sticky, cleared by a Push.	x
31-27		Reserved	

User-Defined Status Registers (USTATx)

These are non-memory mapped, universal, system registers (UREG and SREG). The reset value for these registers is 0x0000 0000. The USTATx registers are user-defined, general-purpose status registers. Programs can use these 32-bit registers with bitwise instructions (set, clear, test, and others). Often, programs use these registers for low-overhead, general-purpose flags or for temporary 32-bit storage of data.

Processing Element Registers

Except for the PX register, the DSP's processing element registers store data for each element's ALU, multiplier, and shifter. The inputs and outputs for processing element operations go through these registers. The PX register lets programs transfer data between the data buses, but cannot be an input or output in a calculation.

Table A-6. Processing Element Universal Registers (UREG)

Register Name and Page Reference	Initialization After Reset
"Data File Data Registers (Rx, Fx, Sx)" on page A-15	Undefined
"Multiplier Results Registers (MRxF, MRxB)" on page A-16	Undefined
"Program Memory Bus Exchange Register (PX)" on page A-16	Undefined

Data File Data Registers (Rx, Fx, Sx)

These are non-memory mapped, universal, data registers (UREG and DREG). Each of the DSP's processing elements has a data register file—a set of 40-bit data registers that transfer data between the data buses and the computation units. These registers also provides local storage for operands and results.

The R, F, and S prefixes on register names do not effect the 32-bit or 40-bit data transfer; the naming convention determines how the ALU, multiplier, and shifter treat the data and determines which processing element's data registers are being used.

For more information on how to use these registers, see "Data Register File" on page 2-29.

Multiplier Results Registers (MRxF, MRxB)

These are non-memory mapped, universal, data registers (UREG and DREG). Each of the DSP's multipliers has a primary or foreground (MRF) register and alternate or background (MRB) results register. Fixed-point operations place 80-bit results in the multiplier's foreground MRF register or background MRB register, depending on which is active.

For more information on selecting the result register, see "Alternate (Secondary) Data Registers" on page -31. For more information on result register fields, see "Data Register File" on page -29.

Program Memory Bus Exchange Register (PX)

These are non-memory mapped, universal registers (UREG only). The PM Bus Exchange (PX) register permits data to flow between the PM and DM data buses. The PX register can work as one 64-bit register or as two 32-bit registers (PX1 and PX2).

For more information on data alignment of PX1 and PX2 within PX and PX register usage, see "Internal Data Bus Exchange" on page -7.

Program Sequencer Registers

The DSP's program sequencer registers direct the execution of instructions. These registers include support for:

- Instruction pipeline
- Program and loop stacks
- Timer
- Interrupt mask and latch

Table A-7. Program Sequencer System Registers (UREG and SREG)

Register	Initialization After Reset
"Interrupt Latch Register (IRPTL)" on page A-18	0x0000 0000 (cleared)
"Interrupt Mask Register (IMASK)" on page A-22	0x0000 0003
"Interrupt Mask Pointer Register (IMASKP)" on page A-23	0x0000 0000 (cleared)
"Link Port Interrupt Register (LIRPTL)" on page A-24	0x0000 0000 (cleared)
"Flag Value Register (FLAGS)" on page A-27	0x0000 000n[1]

1 FLAGS bits 0-3 are equal to the values of the FLAG0-3 input pins after reset; the flag pins are configured as inputs after reset.

Table A-8. Program Sequencer Universal Registers (UREG only)

Register	Initialization After Reset
"Program Counter Register (PC)" on page A-28	Undefined
"Program Counter Stack Register (PCSTK)" on page A-29	Undefined
"Program Counter Stack Pointer Register (PCSTKP)" on page A-29	Undefined
"Fetch Address Register (FADDR)" on page A-30	Undefined
"Decode Address Register (DADDR)" on page A-30	Undefined
"Loop Address Stack Register (LADDR)" on page A-30	Undefined

Table A-8. Program Sequencer Universal Registers (UREG only) (Cont'd)

Register	Initialization After Reset
"Current Loop Counter Register (CURLCNTR)" on page A-31	Undefined
"Loop Counter Register (LCNTR)" on page A-31	Undefined
"Timer Period Register (TPERIOD)" on page A-31	Undefined
"Timer Count Register (TCOUNT)" on page A-31	Undefined

Interrupt Latch Register (IRPTL)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for this register is 0x0000 0000. The IRPTL register indicates latch status for interrupts.

Table A-9. Interrupt Latch Register (IRPTL) Bit Definitions

Bit(s)	Name	Definition
0	EMUI	**Emulator Interrupt.** This bit indicates whether an EMUI interrupt is latched and is pending (if set, =1) or no EMUI interrupt is pending (if cleared, =0). An EMUI interrupt occurs on reset and when an external device asserts the $\overline{\text{EMU}}$ pin.
1	RSTI	**Reset Interrupt.** This bit indicates whether an RSTI interrupt is latched and is pending (if set, =1) or no RSTI interrupt is pending (if cleared, =0). An RSTI interrupt occurs on reset as an external device asserts the $\overline{\text{RESET}}$ pin.
2	IICDI	**Illegal Input Condition Detected.** This bit indicates whether an IICD interrupt is latched and is pending (if set, =1) or no IICD interrupt is pending (if cleared, =0). An IICD interrupt occurs when a TRUE results from the logical Or'ing of the Illegal I/O Processor Register Access (IRA) and Unaligned 64-bit Memory Access bits in the STKYx registers. For more information, see "IRA" on page A-14 and "U64MA" on page A-14.

Table A-9. Interrupt Latch Register (IRPTL) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
3	SOVFI	**Stack Overflow/Full.** This bit indicates whether an SOVFI interrupt is latched and is pending (if set, =1) or no SOVFI interrupt is pending (if cleared, =0). An SOVFI interrupt occurs when a stack in the program sequencer overflows or is full. For more information see "PCFL" on page A-14, "SSOV" on page A-14, and "LSOV" on page A-14.
4	TMZHI	**Timer Expired High Priority.** This bit indicates whether a TMZHI interrupt is latched and is pending (if set, =1) or no TMZHI interrupt is pending (if cleared, =0). A TMZHI interrupt occurs when the timer decrements to zero. Note that this event also triggers a TMZLI interrupt. The following control timer operations: • The TCOUNT register contains the timer counter. The timer decrements the TCOUNT register each clock cycle. • The TPERIOD value specifies the frequency of timer interrupts. The number of cycles between interrupts is TPERIOD + 1. The maximum value of TPERIOD is $2^{32} - 1$. • The TIMEN bit in the MODE2 register starts and stops the timer. Because the timer expired event (TCOUNT decrements to zero) generates two interrupts, TMZHI and TMZLI, programs should unmask the timer interrupt with the desired priority and leave the other one masked.
5	VIRPTI	**Multiprocessor Vector Interrupt.** This bit indicates whether a VIRPTI interrupt is latched and is pending (if set, =1) or no VIRPTI interrupt is pending (if cleared, =0). A VIRPTI interrupt occurs when one of the DSPs in a multiprocessor system writes an address (the vector) to the DSP's VIRPT register.
6	IRQ2I	$\overline{\texttt{IRQ2}}$ Hardware Interrupt. This bit indicates whether an IRQ2I interrupt is latched and is pending (if set, =1) or no IRQ2I interrupt is pending (if cleared, =0). An IRQ2I interrupt occurs when an external device asserts the $\overline{\texttt{IRQ2}}$ pin.
7	IRQ1I	$\overline{\texttt{IRQ1}}$ Hardware Interrupt. This bit indicates whether an IRQ1I interrupt is latched and is pending (if set, =1) or no IRQ1I interrupt is pending (if cleared, =0). An IRQ1I interrupt occurs when an external device asserts the $\overline{\texttt{IRQ1}}$ pin.

Table A-9. Interrupt Latch Register (IRPTL) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
9	IRQ0I	$\overline{\text{IRQ0}}$ Hardware Interrupt. This bit indicates whether an IRQ0I interrupt is latched and is pending (if set, =1) or no IRQ0I interrupt is pending (if cleared, =0). An IRQ0I interrupt occurs when an external device asserts the $\overline{\text{IRQ0}}$ pin.
9		Reserved
10	SPR0I	**SPORT Receive 0.** This bit indicates whether a SPR0I interrupt is latched and is pending (if set, =1) or no SPR0I interrupt is pending (if cleared, =0). A SPR0I interrupt occurs two cycles after the last bit of an input the serial word is latched into RX0.
11	SPR1I	**SPORT Receive 1.** This bit indicates whether a SPR1I interrupt is latched and is pending (if set, =1) or no SPR1I interrupt is pending (if cleared, =0). A SPR1I interrupt occurs two cycles after the last bit of an input the serial word is latched into RX1.
12	SPT0I	**SPORT Transmit 0.** This bit indicates whether a SPT0I interrupt is latched and is pending (if set, =1) or no SPT0I interrupt is pending (if cleared, =0). An SPT0I interrupt occurs two cycles after the last bit of an output the serial word is latched from TX0.
13	SPT1I	**SPORT Transmit 1.** This bit indicates whether a SPT1I interrupt is latched and is pending (if set, =1) or no SPT1I interrupt is pending (if cleared, =0). An SPT1I interrupt occurs two cycles after the last bit of an output the serial word is latched from TX1.
14	LPISUMI	**Link Buffer DMA Summary.** This bit indicates whether an LPISUMI interrupt is latched and is pending (if set, =1) or no LPISUMI interrupt is pending (if cleared, =0). An LPISUMI interrupt occurs when a TRUE results from the logical Or'ing of unmasked link port interrupts, which are configured in the LIRPTL register.
15	EP0I	**External Port Buffer 0 DMA.** This bit indicates whether an EP0I interrupt is latched and is pending (if set, =1) or no EP0I interrupt is pending (if cleared, =0). An EP0I interrupt occurs when the external port buffer's DMA is disabled (DEN=0) and either: • The buffer set to receive (TRAN=0), and the buffer is not empty • The buffer set to transmit (TRAN=1), and the buffer is not full

Table A-9. Interrupt Latch Register (IRPTL) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
16	EP1I	**External Port Buffer 1 DMA.** This bit indicates whether an EP1I interrupt is latched and is pending (if set, =1) or no EP1I interrupt is pending (if cleared, =0). For more information, see "EP0I" on page A-20.
17	EP2I	**External Port Buffer 2 DMA.** This bit indicates whether an EP2I interrupt is latched and is pending (if set, =1) or no EP2I interrupt is pending (if cleared, =0). For more information, see "EP0I" on page A-20.
18	EP3I	**External Port Buffer 3 DMA.** This bit indicates whether an EP3I interrupt is latched and is pending (if set, =1) or no EP3I interrupt is pending (if cleared, =0). For more information, see "EP0I" on page A-20.
19	LSRQI	**Link Port Service Request.** This bit indicates whether an LSRQI interrupt is latched and is pending (if set, =1) or no LSRQI interrupt is pending (if cleared, =0). An LSRQI interrupt occurs when an external source accesses an unassigned link port or accesses an assigned link port that has its link buffer disabled.
20	CB7I	**DAG1 Circular Buffer 7 Overflow.** This bit indicates whether a CB7I interrupt is latched and is pending (if set, =1) or no CB7I interrupt is pending (if cleared, =0). For more information, see "CB7S" on page A-14.
21	CB15I	**DAG2 Circular Buffer 15 Overflow.** This bit indicates whether a CB15I interrupt is latched and is pending (if set, =1) or no CB15I interrupt is pending (if cleared, =0). For more information, see "CB15S" on page A-14.
22	TMZLI	**Timer Expired (Low Priority).** This bit indicates whether a TMZLI interrupt is latched and is pending (if set, =1) or no TMZLI interrupt is pending (if cleared, =0). For more information, see "TMZHI" on page A-19.
23	FIXI	**Fixed-Point Overflow.** This bit indicates whether a FIXI interrupt is latched and is pending (if set, =1) or no FIXI interrupt is pending (if cleared, =0). For more information, see "AOS" on page A-13.

Table A-9. Interrupt Latch Register (IRPTL) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
24	FLTOI	**Floating-Point Overflow.** This bit indicates whether a FLTOI interrupt is latched and is pending (if set, =1) or no FLTOI interrupt is pending (if cleared, =0). For more information, see "AVS" on page A-13.
25	FLTUI	**Floating-Point Underflow.** This bit indicates whether a FLTUI interrupt is latched and is pending (if set, =1) or no FLTUI interrupt is pending (if cleared, =0). For more information, see "AUS" on page A-13.
26	FLTII	**Floating-Point Invalid Operation.** This bit indicates whether a FLTII interrupt is latched and is pending (if set, =1) or no FLTII interrupt is pending (if cleared, =0). For more information, see "AIS" on page A-13.
27	SFT0I	**User Software Interrupt 0.** This bit indicates whether a SFT0I interrupt is latched and is pending (if set, =1) or no SFT0I interrupt is pending (if cleared, =0). An SFT0I interrupt occurs when a program sets (=1) this bit.
28	SFT1I	**User Software Interrupt 1.** This bit indicates whether a SFT1I interrupt is latched and is pending (if set, =1) or no SFT1I interrupt is pending (if cleared, =0). For more information, see "SFT0I" on page A-22.
29	SFT2I	**User Software Interrupt 2.** This bit indicates whether a SFT2I interrupt is latched and is pending (if set, =1) or no SFT2I interrupt is pending (if cleared, =0). For more information, see "SFT0I" on page A-22.
30	SFT3I	**User Software Interrupt 3.** This bit indicates whether a SFT3I interrupt is latched and is pending (if set, =1) or no SFT3I interrupt is pending (if cleared, =0). For more information, see "SFT0I" on page A-22.
31		Reserved

Interrupt Mask Register (IMASK)

This is a non-memory mapped, universal, system register (`UREG` and `SREG`). The reset value for this register is `0x0000 0003`. Each bit in the `IMASK` reg-

ister corresponds to a bit with the same name in the IRPTL registers. The bits in IMASK unmask (enable if set, =1) or mask (disable if cleared, =0) the interrupts that are latched in the IRPTL register. Except for $\overline{\text{RESET}}$, all interrupts are maskable.

When IMASK masks an interrupt, the masking disables the DSP's response to the interrupt. The IRPTL register still latches an interrupt even when masked, and the DSP responds to that latched interrupt if it is later unmasked.

Interrupt Mask Pointer Register (IMASKP)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for this register is 0x0000 0000. Each bit in the IMASKP register corresponds to a bit with the same name in the IRPTL registers. This register supports an interrupt nesting scheme that lets higher priority events interrupt an interrupt service routine and keeps lower priority events from interrupting.

When interrupt nesting is enabled (NESTM=1 in the MODE1 register), the bits in IMASKP mask lower priority and unmask higher priority interrupts than the interrupt that is currently being serviced. The IRPTL register still latches a lower priority interrupt even when masked, and the DSP responds to that latched interrupt if it is later unmasked.

When interrupt nesting is disabled (NESTM=0 in the MODE1 register), the bits in IMASKP mask all interrupts while an interrupt is currently being serviced. The IRPTL register still latches these interrupts even when masked, and the DSP responds to the highest priority latched interrupt after servicing the current interrupt.

For more information, see "NESTM" on page A-4.

Link Port Interrupt Register (LIRPTL)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for these registers is 0x0000 0000. The LIRPTL register indicates latch status, select masking, and displays mask pointers for link port interrupts.

Note that the LPISUM bit in the IRPTL register contains a logical Oring of the link port latch bits (LIRPTL5-0). For more information, see "LPISUMI" on page A-20.

Table A-10. Link Port Interrupt Latch, Mask, and Mask Pointer Register (LIRPTL) Bit Definitions

Bit	Name	Definition
0	LP0	**Link Port Buffer 0 DMA.** This bit indicates whether an LP0 interrupt is latched and is pending (if set, =1) or no LP0 interrupt is pending (if cleared, =0). An LP0 interrupt occurs when the link port buffer's DMA is disabled (DEN=0) and either: • The buffer set to receive (TRAN=0), and the buffer is not empty • The buffer set to transmit (TRAN=1), and the buffer is not full
1	LP1	**Link Port Buffer 1 DMA.** This bit indicates whether an LP1 interrupt is latched and is pending (if set, =1) or no LP1 interrupt is pending (if cleared, =0). For more information, see "LP0" on page A-24.
2	LP2	**Link Port Buffer 2 DMA.** This bit indicates whether an LP2 interrupt is latched and is pending (if set, =1) or no LP2 interrupt is pending (if cleared, =0). For more information, see "LP0" on page A-24.
3	LP3	**Link Port Buffer 3 DMA.** This bit indicates whether an LP3 interrupt is latched and is pending (if set, =1) or no LP3 interrupt is pending (if cleared, =0). For more information, see "LP0" on page A-24.
4	LP4	**Link Port Buffer 4 DMA.** This bit indicates whether an LP4 interrupt is latched and is pending (if set, =1) or no LP4 interrupt is pending (if cleared, =0). For more information, see "LP0" on page A-24.

Table A-10. Link Port Interrupt Latch, Mask, and Mask Pointer Register (LIRPTL) Bit Definitions (Cont'd)

Bit	Name	Definition
5	LP5	**Link Port Buffer 5 DMA.** This bit indicates whether an LP5 interrupt is latched and is pending (if set, =1) or no LP5 interrupt is pending (if cleared, =0). For more information, see "LP0" on page A-24.
15-6		Reserved
16	LP0MSK	**Link Buffer 0 DMA Interrupt Mask.** This bit unmasks the LP0 interrupt (if set, =1) or masks the LP0 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
17	LP1MSK	**Link Buffer 1 DMA Interrupt Mask.** This bit unmasks the LP1 interrupt (if set, =1) or masks the LP1 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
18	LP2MSK	**Link Buffer 2 DMA Interrupt Mask.** This bit unmasks the LP2 interrupt (if set, =1) or masks the LP2 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
19	LP3MSK	**Link Buffer 3 DMA Interrupt Mask.** This bit unmasks the LP3 interrupt (if set, =1) or masks the LP3 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
20	LP4MSK	**Link Buffer 4 DMA Interrupt Mask.** This bit unmasks the LP4 interrupt (if set, =1) or masks the LP4 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
21	LP5MSK	**Link Buffer 5 DMA Interrupt Mask.** This bit unmasks the LP5 interrupt (if set, =1) or masks the LP5 interrupt (if cleared, =0). For more information on how interrupt masking works, see "Interrupt Mask Register (IMASK)" on page A-22.
23-22		Reserved

Table A-10. Link Port Interrupt Latch, Mask, and Mask Pointer Register (LIRPTL) Bit Definitions (Cont'd)

Bit	Name	Definition
24	LP0MSKP	**Link Buffer 0 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP0 interrupt is masked (if set, =1) or the LP0 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.
25	LP1MSKP	**Link Buffer 1 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP1 interrupt is masked (if set, =1) or the LP1 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.
26	LP2MSKP	**Link Buffer 2 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP2 interrupt is masked (if set, =1) or the LP2 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.
27	LP3MSKP	**Link Buffer 3 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP3 interrupt is masked (if set, =1) or the LP3 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.
28	LP4MSKP	**Link Buffer 4 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP4 interrupt is masked (if set, =1) or the LP4 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.

Table A-10. Link Port Interrupt Latch, Mask, and Mask Pointer Register (LIRPTL) Bit Definitions (Cont'd)

Bit	Name	Definition
29	LP5MSKP	**Link Buffer 5 DMA Interrupt Mask Pointer.** When the DSP is servicing another interrupt, this bit indicates whether the LP5 interrupt is masked (if set, =1) or the LP5 interrupt is unmasked (if cleared, =0). For more information on how interrupt mask pointers works, see "Interrupt Mask Pointer Register (IMASKP)" on page A-23.
31-30		Reserved

Flag Value Register (FLAGS)

This is a non-memory mapped, universal, system register (UREG and SREG). The reset value for these registers is 0x0000 0000. The FLAGS register indicates the state of the FLGx pins. When a FLGx pin is an output, the DSP outputs a high when a program sets the pin's bit in FLAGS. The I/O direction (input or output) selection of each bit is controlled by its FLGxO bit in the MODE2 register. For more information, see "FLG0O" on page A-7.

Table A-11. Link Port Interrupt Latch, Mask, and Mask Pointer Register (LIRPTL) Bit Definitions

Bit	Name	Definition
0	FLG0	**FLAG0 Value.** This bit indicates the state of the FLAG0 pin, whether the pin is high (if set, =1) or low (if cleared, =0).
1	FLG1	**FLAG1 Value.** This bit indicates the state of the FLAG1 pin, whether the pin is high (if set, =1) or low (if cleared, =0).
2	FLG2	**FLAG2 Value.** This bit indicates the state of the FLAG2 pin, whether the pin is high (if set, =1) or low (if cleared, =0).
3	FLG3	**FLAG3 Value.** This bit indicates the state of the FLAG3 pin, whether the pin is high (if set, =1) or low (if cleared, =0).
31-4		Reserved

Program Counter Register (PC)

This is a non-memory mapped, universal register (UREG only). The Program Counter register is the last stage in the fetch-decode-execute instruction pipeline and contains the 24-bit address of the instruction that the DSP executes on the next cycle. The PC couples with the Program Counter Stack, PCSTK, which stores return addresses and top-of-loop addresses. All addresses generated by the sequencer are 24-bit program memory instruction addresses.

As shown in Figure A-1, the address buses can handle 32-bit addresses, but the program sequencer only generates 24-bit addresses over the PM bus. Because the sequencer generates 24-bit addresses, sequencing is limited to the low 12 Mwords of the DSP's 4 Gword memory map.

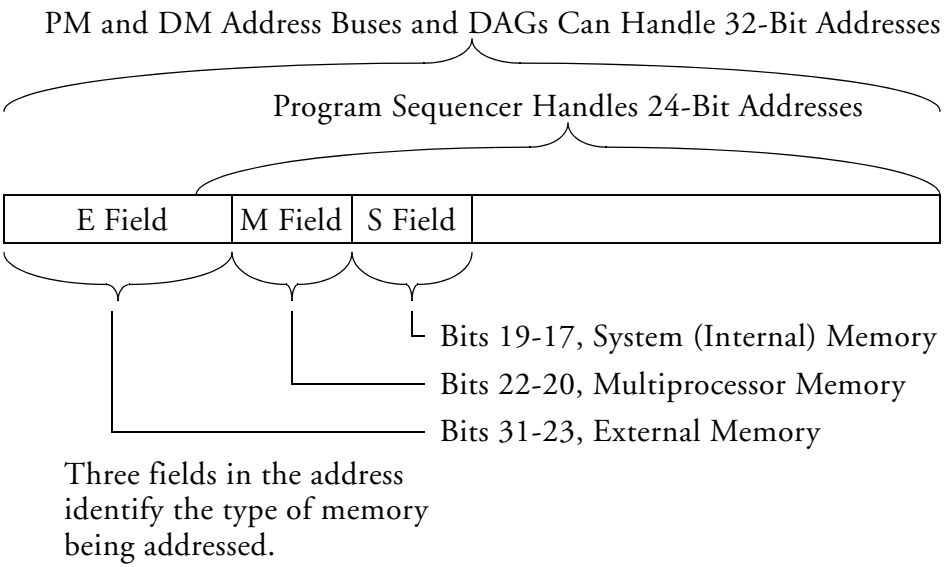

PM and DM Address Buses and DAGs Can Handle 32-Bit Addresses
Program Sequencer Handles 24-Bit Addresses
E Field
M Field
S Field
Bits 19-17, System (Internal) Memory
Bits 22-20, Multiprocessor Memory
Bits 31-23, External Memory
Three fields in the address identify the type of memory being addressed.

Figure A-1. PM and DM Bus Addresses Versus Sequencing Addresses

Table A-12 describes the three fields that appear in Figure A-1. The content of the External (E), Multiprocessor (M), and System (S) fields in the address route the data or instruction access to the memory space.

Table A-12. PM and DM Address Bus E, M, and S Fields

Bit Field	Description
E	External — Values in this field have the following meaning: **non-zero:** the address is in external memory; with the E bits active remaining bits [22-0] are a valid address. **all zeros:** the address is in the DSP's internal memory or in the internal memory of another ADSP-21160 DSP (M and S activated).
M	Multiprocessor — Values in this field have the following meaning: **non-zero:** ID of another ADSP-21160 **111:** broadcast write to internal memory of all ADSP-21160s **000:** address in the DSP's own internal memory
S	System — Values in this field have the following meaning: **000:** address of an IOP register **001:** address in Long Word Addressing space **01x:** address in Normal Word Addressing space **1xx:** address in Short Word Addressing space

Program Counter Stack Register (PCSTK)

This is a non-memory mapped, universal register (UREG only). The Program Counter Stack register contains the address of the top of the PC stack. This register is a readable and writable register.

Program Counter Stack Pointer Register (PCSTKP)

This is a non-memory mapped, universal register (UREG only). The Program Counter Stack Pointer register contains the value of PCSTKP. This value is zero when the PC stack is empty, is 1...30 when the stack contains data, and is 31 when the stack overflows. This register is readable and writable. A write to PCSTKP takes effect after a one-cycle delay. If the PC stack is overflowed, a write to PCSTKP has no effect.

Fetch Address Register (FADDR)

This is a non-memory mapped, universal register (UREG only). The Fetch Address register is the first stage in the fetch-decode-execute instruction pipeline and contains the 24-bit address of the instruction that the DSP fetches from memory on the next cycle.

Decode Address Register (DADDR)

This is a non-memory mapped, universal register (UREG only). The Decode Address register is the second stage in the fetch-decode-execute instruction pipeline and contains the 24-bit address of the instruction that the DSP decodes on the next cycle.

Loop Address Stack Register (LADDR)

This is a non-memory mapped, universal register (UREG only). The Loop Address Stack is six levels deep by 32 bits wide. The 32-bit word of each level consists of a 24-bit loop termination address, a 5-bit termination code, and a 2-bit loop type code:

Table A-13. Loop Address Stack Register (LADDR)

Bits	Value
0-23	loop termination address
24-28	termination code
29	reserved (always reads 0)
30-31	loop type code 00 = arithmetic condition-based (not LCE) 01 = counter-based, length 1 10 = counter-based, length 2 11 = counter-based, length > 2

Current Loop Counter Register (CURLCNTR)

This is a non-memory mapped, universal register (UREG only). The Current Loop Counter register provides access to the loop counter stack and tracks iterations for the Do/Until LCE loop being executed.

For more information on how to use CURLCNTR, see "Loop Counter Stack" on page -30.

Loop Counter Register (LCNTR)

This is a non-memory mapped, universal register (UREG only). The Loop Counter register provides access to the loop counter stack and holds the count value before the Do/Until LCE loop is executed. For more information on how to use LCNTR, see "Loop Counter Stack" on page -30.

Timer Period Register (TPERIOD)

This is a non-memory mapped, universal register (UREG only). The Timer Period register contains the decrementing timer count value, counting down the cycles between timer interrupts. For more information on how to use the timer, see "Timer and Sequencing" on page -50.

Timer Count Register (TCOUNT)

This is a non-memory mapped, universal register (UREG only). The Timer Count register contains the timer period, indicating the number of cycles between timer interrupts. For more information on how to use the timer, see "Timer and Sequencing" on page -50.

Data Address Generator Registers

The DSP's Data Address Generator (DAG) registers hold data addresses, modify values, and circular buffer configurations. Using these registers,

the DAGs can automatically increment addressing for ranges of data locations (a buffer).

Table A-14. Data Address Generator Universal Registers (UREG only)

Register	Initialization After Reset
"Index Registers (Ix)" on page A-32	Undefined
"Modify Registers (Mx)" on page A-32	Undefined
"Length and Base Register (Lx, Bx)" on page A-33	Undefined

Index Registers (Ix)

These are non-memory mapped, universal registers (`UREG` only). The Data Address Generators store addresses in Index registers (`I0-I7` for DAG1 and `I8-I15` for DAG2). An index register holds an address and acts as a pointer to memory.

For more information, see "Overview" in Chapter 4, Data Address Generators.

Modify Registers (Mx)

These are non-memory mapped, universal registers (`UREG` only). The Data Address Generators update stored addresses using Modify registers (`M0-M7` for DAG1 and `M8-M15` for DAG2). A modify register provides the increment or step size by which an index register is pre- or post-modified during a register move.

For more information, see "Overview" in Chapter 4, Data Address Generators.

Length and Base Register (Lx, Bx)

These are non-memory mapped, universal registers (UREG only). The Data Address Generators control circular buffering operations with Length and Base registers (L0-L7 and B0-B7 for DAG1 and L8-L15 and B8-B15 for DAG2). Length and base registers setup the range of addresses and the starting address for a circular buffer.

For more information, see "Overview" in Chapter 4, Data Address Generators.

I/O Processor Registers

The I/O processor's registers are accessible as part of the DSP's memory map. Table A-16 on page A-36 lists the I/O processor's memory mapped registers in address order and provides a cross reference to a description of each register. These registers occupy addresses 0x00 through 0xFF of the memory map and control I/O operations, including:

- External port DMA
- Link port DMA
- Serial port DMA



I/O processor registers have a one cycle effect latency (changes take effect on the second cycle after the change).

Because the I/O processor's registers are part of the DSP's memory map, buses access these registers as locations in memory. While these registers act as memory mapped locations, they are separate from the DSP's internal memory and have different bus access. One bus can access one I/O processor register from one I/O processor register group at a time. Table A-15 lists the I/O processor register groups.

When there is contention among the buses for access to registers in the same I/O processor register group, the DSP arbitrates register access as follows:

- External Port (EP) bus accesses (highest priority)
- Data Memory (DM) bus
- Program Memory (PM) bus
- I/O processor (IO) bus (lowest priority)

The bus with highest priority gets access to the I/O processor register group, and the other buses are held off from accessing that I/O processor register group until that access been completed.

There is one exception to this access contention rule. The IO bus and EP bus can simultaneously access the DB (DMA buffer) group of registers, allowing DMA transfers to internal memory at full speed.

Table A-15. I/O Processor Register Groups

Register Group	I/O Processor Registers In This Group
System Control (SC) Registers	SYSCON, VIRPT, WAIT, SYSTAT, MSGR0, MSGR1, MSGR2, MSGR3, MSGR4, MSGR5, MSGR6, MSGR7, BMAX, BCNT, ELAST, PC_SHDW, MODE2_SHDW
DMA Address (DA) Registers	II4, IM4, C4, CP4, GP4, DB4, DA4, II5, IM5, C5, CP5, GP5, DB5, DA5, II6, IM6, C6, CP6, GP6, EI6, EM6, EC6, II7, IM7, C7, CP7, GP7, EI7, EM7, EC7, II8, IM8, C8, CP8, GP8, EI8, EM8, EC8, II9, IM9, C9, CP9, GP9, EI9, EM9, EC9, II0, IM0, C0, CP0, GP0, DB0, DA0, II1, IM1, C1, CP1, GP1, DB1, DA1, II2, IM2, C2, CP2, GP2, DB2, DA2, II3, IM3, C3, CP3, GP3, DB3, DA3, DMASTAT

Table A-15. I/O Processor Register Groups

Register Group	I/O Processor Registers In This Group
DMA Buffer (DB) Registers	EPB0, EPB1, EPB2, EPB3, DMAC6, DMAC7, DMAC8, DMAC9
Link and Serial Port (LSP) Registers	LBUF0, LBUF1, LBUF2, LBUF3, LBUF4, LBUF5, LCTL, LCOM, LAR, LSRQ, LPATH1, LPATH2, LPATH3, LPCNT, CNST1, CNST2, STCTL0, SRCTL0, TX0, RX0, TDIV0, RDIV0, MTCS0, MRCS0, MTCCS0, MRCCS0, SPATH0, KEYWD0, KEYMASK0, STCTL1, SRCTL1, TX1, RX1, TDIV1, RDIV1, MTCS1, MRCS1, MTCCS1, MRCCS1, SPATH1, KEYWD1, KEYMASK1

Because the I/O processor registers are memory-mapped, the DSP's architecture does not allow programs to directly transfer data between these registers and other memory locations, except as part of a DMA operation. To read or write I/O processor registers, programs must use the processor core registers. The following example code shows a value being transferred from memory to the `USTAT1` register, then the value is transferred to the I/O processor `WAIT` registers.

```
USTAT2= 0x108421; /* 1st instr. to be executed after reset */
DM(WAIT)=USTAT2; /* Set external memory waitstates to 0   */
```

The register names for I/O processor registers are not part of the DSP's assembly syntax. To ease access to these registers, programs should use the `#include` command to incorporate a file containing the registers' symbolic

names and addresses. An example `#include` file appears in the "Register and Bit #Defines File (def21160.h)" on page A-82.

Table A-16. I/O Processor Registers Memory Map

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x00	SYSCON	0x0001 0010	SC	on page A-45
0x01	VIRPT	0x0004 0014	SC	on page A-47
0x02	WAIT	0x01ce 739c	SC	on page A-48
0x03	SYSTAT	0x000 0nn0	SC	on page A-51
0x04	EPB0	ni	DB	on page A-52
0x06	EPB1	ni	DB	on page A-52
0x08	MSGR0	ni	SC	on page A-53
0x09	MSGR1	ni	SC	on page A-53
0x0a	MSGR2	ni	SC	on page A-53
0x0b	MSGR3	ni	SC	on page A-53
0x0c	MSGR4	ni	SC	on page A-53
0x0d	MSGR5	ni	SC	on page A-53
0x0e	MSGR6	ni	SC	on page A-53
0x0f	MSGR7	ni	SC	on page A-53
0x10	PC_SHDW	ni	SC	on page A-53
0x11	MODE2_ SHDW	ni	SC	on page A-53
0x12 – 0x13	Reserved			Reserved
0x14	EPB2	ni	DB	on page A-52
0x16	EPB3	ni	DB	on page A-52

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x18	BMAX	0x0000 0000	SC	on page A-53
0x19	BCNT	0x0000 0000	SC	on page A-54
0x1a	Reserved			Reserved
0x1b	ELAST	ni	SC	on page A-54
0x1c	DMAC10	ni*	DB	on page A-54
0x1d	DMAC11	0x0000 0000	DB	on page A-54
0x1e	DMAC12	0x0000 0000	DB	on page A-54
0x1f	DMAC13	0x0000 0000	DB	on page A-54
0x20 – 0x2f	Reserved			Reserved
0x30	II4	ni	DA	on page A-58
0x31	IM4	ni	DA	on page A-59
0x32	C4	ni	DA	on page A-59
0x33	CP4	ni	DA	on page A-59
0x34	GP4	ni	DA	on page A-60
0x35	DB4	ni	DA	on page A-60
0x36	DA4	ni	DA	on page A-60
0x37	DMASTAT	ni	DA	on page A-60
0x38	II5	ni	DA	on page A-58
0x39	IM5	ni	DA	on page A-59
0x3a	C5	ni	DA	on page A-59
0x3b	CP5	ni	DA	on page A-59
0x3c	GP5	ni	DA	on page A-60

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x3d	DB5	ni	DA	on page A-60
0x3e	DA5	ni	DA	on page A-60
0x3f	Reserved	ni	DA	Reserved
0x40	II10	ni*	DA	on page A-58
0x41	IM10	ni*	DA	on page A-59
0x42	C10	ni*	DA	on page A-59
0x43	CP10	ni*	DA	on page A-59
0x44	GP10	ni*	DA	on page A-60
0x45	EI10	ni*	DA	on page A-61
0x46	EM10	ni*	DA	on page A-61
0x47	EC10	ni*	DA	on page A-62
0x48	II11	ni	DA	on page A-58
0x49	IM11	ni	DA	on page A-59
0x4a	C11	ni	DA	on page A-59
0x4b	CP11	ni	DA	on page A-59
0x4c	GP11	ni	DA	on page A-60
0x4d	EI11	ni	DA	on page A-61
0x4e	EM11	ni	DA	on page A-61
0x4f	EC11	ni	DA	on page A-62
0x50	II12	ni	DA	on page A-58
0x51	IM12	ni	DA	on page A-59
0x52	C12	ni	DA	on page A-59

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x53	CP12	ni	DA	on page A-59
0x54	GP12	ni	DA	on page A-60
0x55	EI12	ni	DA	on page A-61
0x56	EM12	ni	DA	on page A-61
0x57	EC12	ni	DA	on page A-62
0x58	II13	ni	DA	on page A-58
0x59	IM13	ni	DA	on page A-59
0x5a	C13	ni	DA	on page A-59
0x5b	CP13	ni	DA	on page A-59
0x5c	GP13	ni	DA	on page A-60
0x5d	EI13	ni	DA	on page A-61
0x5e	EM13	ni	DA	on page A-61
0x5f	EC13	ni	DA	on page A-62
0x60	II0	ni	DA	on page A-58
0x61	IM0	ni	DA	on page A-59
0x62	C0	ni	DA	on page A-59
0x63	CP0	ni	DA	on page A-59
0x64	GP0	ni	DA	on page A-60
0x65	DB0	ni	DA	on page A-60
0x66	DA0	ni	DA	on page A-60
0x68	II1	ni	DA	on page A-58
0x69	IM1	ni	DA	on page A-59

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x6a	C1	ni	DA	on page A-59
0x6b	CP1	ni	DA	on page A-59
0x6c	GP1	ni	DA	on page A-60
0x6d	DB1	ni	DA	on page A-60
0x6e	DA1	ni	DA	on page A-60
0x6f	Reserved			Reserved
0x70	II2	ni	DA	on page A-58
0x71	IM2	ni	DA	on page A-59
0x72	C2	ni	DA	on page A-59
0x73	CP2	ni	DA	on page A-59
0x74	GP2	ni	DA	on page A-60
0x75	DB2	ni	DA	on page A-60
0x76	DA2	ni	DA	on page A-60
0x78	II3	ni	DA	on page A-58
0x79	IM3	ni	DA	on page A-59
0x7a	C3	ni	DA	on page A-59
0x7b	CP3	ni	DA	on page A-59
0x7c	GP3	ni	DA	on page A-60
0x7d	DB3	ni	DA	on page A-60
0x7e	DA3	ni	DA	on page A-60
0x7f	Reserved			Reserved
0x80	II6	ni	DA	on page A-58

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x81	IM6	ni	DA	on page A-59
0x82	C6	ni	DA	on page A-59
0x83	CP6	ni	DA	on page A-59
0x84	GP6	ni	DA	on page A-60
0x85	DB6	ni	DA	on page A-60
0x86	DA6	ni	DA	on page A-60
0x88	II7	ni	DA	on page A-58
0x89	IM7	ni	DA	on page A-59
0x8a	C7	ni	DA	on page A-59
0x8b	CP7	ni	DA	on page A-59
0x8c	GP7	ni	DA	on page A-60
0x8d	DB7	ni	DA	on page A-60
0x8e	DA7	ni	DA	on page A-60
0x8f	Reserved			Reserved
0x90	II8	ni	DA	on page A-58
0x91	IM8	ni	DA	on page A-59
0x92	C8	ni	DA	on page A-59
0x93	CP8	ni	DA	on page A-59
0x94	GP8	ni	DA	on page A-60
0x95	DB8	ni	DA	on page A-60
0x96	DA8	ni	DA	on page A-60
0x98	II9	ni	DA	on page A-58

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0x99	IM9	ni	DA	on page A-59
0x9a	C9	ni	DA	on page A-59
0x9b	CP9	ni	DA	on page A-59
0x9c	GP9	ni	DA	on page A-60
0x9d	DB9	ni	DA	on page A-60
0x9e	DA9	ni	DA	on page A-60
0x9f–0xbf	Reserved (emulation control registers)			
0xc0	LBUF0	ni	LSP	on page A-62
0xc2	LBUF1	ni	LSP	on page A-62
0xc4	LBUF2	ni	LSP	on page A-62
0xc6	LBUF3	ni	LSP	on page A-62
0xc8	LBUF4	ni	LSP	on page A-62
0xca	LBUF5	ni	LSP	on page A-62
0xcc	LCTL0	0x00000000	LSP	on page A-62
0xcd	LCTL1	0x00000000	LSP	on page A-62
0xce	LCOM	0x00000000	LSP	on page A-65
0xcf	LAR	0x0002C688	LSP	on page A-67
0xd0	LSRQ	0x0000 0000	LSP	on page A-68
0xd1	LPATH1	ni	LSP	on page A-70
0xd2	LPATH2	ni	LSP	on page A-70
0xd3	LPATH3	ni	LSP	on page A-70
0xd4	LPCNT	ni	LSP	on page A-70

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0xd5	CNST1	ni	LSP	on page A-71
0xd6	CNST2	ni	LSP	on page A-71
0xd7 – 0xdf	Reserved			Reserved
0xe0	STCTL0	0x0000 0000	LSP	on page A-71
0xe1	SRCTL0	0x0000 0000	LSP	on page A-73
0xe2	TX0	ni	LSP	on page A-73
0xe3	RX0	ni	LSP	on page A-76
0xe4	TDIV0	ni	LSP	on page A-77
0xe5	TCNT0	ni	LSP	on page A-78
0xe6	RDIV0	ni	LSP	on page A-78
0xe7	RCNT0	ni	LSP	on page A-79
0xe8	MTCS0	ni	LSP	on page A-79
0xe9	MRCS0	ni	LSP	on page A-80
0xea	MTCCS0	ni	LSP	on page A-80
0xeb	MRCCS0	ni	LSP	on page A-80
0xec	KEYWD0	ni	LSP	on page A-81
0xed	KEYMASK0	ni	LSP	on page A-81
0xee	SPATH0	0x0000 0001	LSP	on page A-81
0xef	SPCNT0	0x0000 0001	LSP	on page A-82
0xf0	STCTL1	0x0000 0000	LSP	on page A-71
0xf1	SRCTL1	0x0000 0000	LSP	on page A-73
0xf2	TX1	ni	LSP	on page A-73

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

Table A-16. I/O Processor Registers Memory Map (Cont'd)

Register Address	Register Name	Initialization After Reset	Register Group	Page Cross Reference
0xf3	RX1	ni	LSP	on page A-76
0xf4	TDIV1	ni	LSP	on page A-77
0xf5	TCNT1	ni	LSP	on page A-78
0xf6	RDIV1	ni	LSP	on page A-78
0xf7	RCNT1	ni	LSP	on page A-79
0xf8	MTCS1	ni	LSP	on page A-79
0xf9	MRCS1	ni	LSP	on page A-80
0xfa	MTCCS1	ni	LSP	on page A-80
0xfb	MRCCS1	ni	LSP	on page A-80
0xfc	KEYWD1	ni	LSP	on page A-81
0xfd	KEYMASK1	ni	LSP	on page A-81
0xfe	SPATH1	0x0000 0001	LSP	on page A-81
0xff	SPCNT1	0x0000 0001	LSP	on page A-82

Notes: An "ni" in the Initialization column indicates that the register is Not Initialized.
For information on Register Groups, see Table A-15 on page A-34.
An * denotes that initialization depends on the booting mode. For more information, see "Bootloading Through The External Port" on page 6-76 or "Bootloading Through The Link Port" on page 6-88.

System Configuration Register (SYSCON)

This register's address is 0x00. The reset value for this register is 0x10, configuring the HPM bits for 16-to-32/64 bit packing (see Table A-17).

Table A-17. System Configuration Register (SYSCON) Bit Definitions

Bit(s)	Name	Definition
0	SRST	**Software Reset.** This bit resets (when set, =1) the DSP. When a program sets (=1) SRST, the DSP responds to the non-maskable RSTI interrupt and clears (=0) SRST.
1	BSO	**Boot Select Override.** This bit enables (if set, =1) or disables (if cleared, =0) access to Boot Memory Space. When BSO is set, the DSP uses the $\overline{\text{BMS}}$ select line (instead of $\overline{\text{MS3-0}}$) to perform DMA channel 10 accesses of external memory. The DSP uses 8- to 48-bit packing when reading from 8-bit boot memory space, but does no packing on writes to this space. For appropriate byte alignment on DMA writes to boot memory space, programs must use the shifter to place the ordered bytes for the transfer in bits 39-32 of each internal Long word address.
2	IIVT	**Internal Interrupt Vector Table.** This bit forces placement of the interrupt vector table at address 0x0004 0000 regardless of booting mode (if 1) or allows placement of the interrupt vector table as selected by the booting mode (if 0).
3		Reserved
6-4	HPM	**Host Packing Mode.** These bits select the external bus packing mode for host accesses as follows: 000=no packing, 001=16-to-32/64 (reset value), 010=16-to-48, 011=32-to-48, 100=32-to-32/64
7	HMSWF	**Host Most Significant Word First Packing Select.** This bit selects the word packing order for host accesses as most-significant-word first (if set, =1) or least-significant-word first (if cleared, =0).

Table A-17. System Configuration Register (SYSCON) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
8	HPFLSH	**Host Packing Status Flush.** This bit flushes (when set, =1) settings for the direct write FIFO. Flushing these settings does the following: • Clears (=0) the HPS status bits in the SYSTAT register • Clears (=0) the channel's DMA request counter • Clears (-0) any partially packed words When a program sets (=1) HPFLSH, the DSP flushes the settings and clears (=0) HPFLSH. There is a two-cycle effect latency in completing the flush operation. Programs must not set the buffer's HPFLSH during the same write that enables the buffer. Also, programs must not set the HPFLSH bit while the DMA channel is active. Programs should determine the channel's active status by reading the corresponding bit in the DMASTAT register.
9	IMDW0	**Internal Memory Block 0 Data Width.** This bit selects the Normal word data access size for internal memory Block 0 as 40-bit data (if set, =1) or 32-bit data (if cleared, =0).
10	IMDW1	**Internal Memory Block 1 Data Width.** This bit selects the Normal word data access size for internal memory Block 1 as 40-bit data (if set, =1) or 32-bit data (if cleared, =0).
11	ADREDY	**Active Drive REDY.** This bit selects line driver type for the DSP's REDY pin as active drive (a/d) (if set, =1) or open drain (o/d) (if cleared, =0).
15-12	MSIZE	**Memory Bank Size.** These bits select the size of the four external memory banks (Bank 3-0). The external memory that is not allotted to a bank is part of the Unbanked external memory region. The formula for external memory bank size is: MSIZE=$\log_2$(desired bank size in words) - 13
16	BHD	**Buffer Hang Disable.** This bit controls whether the processor core proceeds (hang disabled if set, =1) or is held-off (hang enabled if cleared, =0) when the core tries to read from an empty EPBx, Tx, or LBUFx buffer or tries to write to a full EPBX, Rx, or LBUFx buffer.

Table A-17. System Configuration Register (SYSCON) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
18-17	EBPR	**External Bus Priority.** These bits select the priority for the I/O processor's EP bus when arbitrating access to the DSP's external port as follows: 00—priority rotates between DM or PM and IO buses, 01—the winning DM or PM bus has priority over the IO bus, 10—the IO bus has priority over the winning DM or PM bus.
19	DCPR	**External Port DMA Channel Priority Rotation Enable.** This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation among external port DMA channels (channel 10-13).
20	LDCPR	**Link Port DMA Channel Priority Rotation Enable.** This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation among link port DMA channels (channel 4-9).
21	PRROT	**Link–External Port DMA Channel Priority Rotation Enable.** This bit enables (rotates if set, =1) or disables (fixed if cleared, =0) priority rotation between link port DMA channels (channel 4-9) and external port DMA channels (channel 10-13).
22	COD	**CLKOUT Disable.** This bit enables (if set, =1) or disables (if cleared, =0) DSP clock output on the CLKOUT pin. If enabled, the DSP outputs the clock signal on CLKOUT. If disabled, the DSP three-states the CLKOUT pin. This bit is the only way to control the CLKOUT pin.
31-23		Reserved

Vector Interrupt Address Register (VIRPT)

This register's address is `0x01`. The reset value for this register is `0x0004 0014` (see Table A-18). The sequencer uses the `VIRPT` register to support multiprocessor vector interrupts. The vector interrupt (`VIRPTI`) permits passing interprocessor commands in multiple-processor systems. This interrupt occurs when an external processor (a host or another DSP)

writes an address to the VIRPT register, inserting a new vector address for VIRPTI.

Table A-18. Vector Interrupt Address Register (VIRPT) Bit Definitions

Bit(s)	Name	Definition
23-0	VIRPTA	**Vector Interrupt Address.** These bits contain the multiprocessor interrupt's vector (address). When an external processor loads an address into this register, the DSP pushes the status stack and starts executing the routine at the vector address.
31-25	VIRPTD	**Vector Interrupt (optional) Data.** These bits contain optional data that the external processor may pass to the interrupt service routine.

External Memory Waitstate and Access Mode Register (WAIT)

This register's address is 0x02. The reset value for this register is 0x01ce739c, which equates to the following DSP external memory settings: **asynchronous access mode** for all external memory banks, **seven waitstates** with a hold cycle for all accesses to external memory banks,

external DRAM page size of 256 words (if installed), and **disable idle cycle** for DMA handshake (see Table A-19).

Table A-19. External Memory Setup Register (WAIT) Bit Definitions

Bit(s)	Name	Definition
1-0	EB0AM	**External Bank 0 Access Mode.** These bits select the access mode for external memory Bank 0 as follows: EBxAM=External Bank Access Mode 00=Asynchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change before CLKOUT's edge—accesses use the waitstate count setting from EBxWS and require external acknowledge (ACK), allowing a de-asserted ACK to extend the access time. 01=Synchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change on CLKOUT's edge—reads use the waitstate count setting from EBxWS (minimum EBxWS=001); writes are 0-wait state. 10=Synchronous—DSP $\overline{\text{RDH/L}}$ and $\overline{\text{WRH/L}}$ strobes change on CLKOUT's edge—reads use the waitstate count setting from EBxWS (minimum EBxWS=001); writes are 1-wait state. 11=Reserved
4-2	EB0WS	**External Bank 0 Waitstates.** These bit fields select the waitstates for external memory Bank 0 as follows: EBxWS# of WaitstatesHold Time Cycle? 0000no 0011no 0102yes 0113yes 1004yes 1015yes 1106yes 1117yes Note that Hold Cycles applies to asynchronous mode only.
6-5	EB1AM	**External Bank 1 Access Mode.** (see EB0AM definition)
9-7	EB1WS	**External Bank 1 Waitstates.** (see EB0WS definition)
11-10	EB2AM	**External Bank 2 Access Mode.** (see EB0AM definition)
14-12	EB2WS	**External Bank 2 Waitstates.** (see EB0WS definition)
16-15	EB3AM	**External Bank 3 Access Mode.** (see EB0AM definition)

Table A-19. External Memory Setup Register (WAIT) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
19-17	EB3WS	**External Bank 3 Waitstates.** (see EB0WS definition)
21-20	UBAM	**External Unbanked Access Mode.** (see EB0AM definition)
24-22	UBWS	**External Unbanked Waitstates.** (see EB0WS definition)
27-25	PAGSZ	**DRAM Page Size.** These bits select the page size for external DRAM (allowed in Bank 0 only) as follows: PAGSZDRAM Page Size 000256 words 001512 words 0101024 words (1K) 0112048 words (2K) 1004096 words (4K) 1018192 words (8K) 11016384 words (16K) 11132768 words (32K)
29-28		Reserved
30	HIDMA	**Handshake and Idle for DMA Enable.** This bit enables (if set, =1) or disables (if cleared, =0) adding an idle cycle after every memory access for DMAs with handshaking ($\overline{\text{DMAR}}$-$\overline{\text{DMAG}}$). The added cycle reduces bus contention by accommodating devices with a slow three-state time. Also, the added cycle accommodates long write recovery time by de-asserting $\overline{\text{DMAG}}$ longer.
31		Reserved

System Status Register (SYSTAT)

This register's address is 0x03. The reset value has all bits initialized to zero, except for the IDC, CRBM, CRAT fields, which are set from values on the DSP's pins (see Table A-20 on page A-51).

Table A-20. System Status Register (SYSTAT) Bit Definitions

Bit(s)	Name	Definition
0	HSTM	**Host Bus Master.** This bit indicates whether the Host processor has control of the external bus (host bus master if set, =1) or does not have control of the bus (host not bus master if cleared, =0, reset value).
1	BSYN	**Bus Synchronized.** This bit indicates when the DSP's bus arbitration logic is synchronized (if set, =1) or is not synchronized (if cleared, =0, reset value).
3-2		Reserved (reset value =0)
6-4	CRBM	**Current Bus Master.** These bits indicate the ID of the DSP that currently is the bus master in a multiprocessor system. Because CRBM is only valid for DSPs with ID inputs other than zero (e.g. a multiprocessor system), the DSP keeps CRBM set to 001 when ID equals 000. The reset value of CRBM is undefined.
7		Reserved (reset value =0)
10-8	IDC	**ID Code.** These bits indicate the state of the ID pins on the DSP. The reset value of IDCID is undefined.
11		Reserved (reset value =0)
12	DWPD	**Direct Write Pending.** This bit indicates when a direct write to DSP's internal memory is pending (if set, =1) or is not pending (if cleared, =0, reset value). The DSP clears DWPD when the direct write is compete. If an external device attempts a direct write during a DMA chaining operation or if higher priority DMA request occurs, the direct write may be delayed for several cycles. The maximum delay for a pending direct write is 12 cycles.

Table A-20. System Status Register (SYSTAT) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
13	VIPD	**Vector Interrupt Pending.** This bit indicates when a vector interrupt is pending (if set, =1) or is not pending (if cleared, =0, reset value). A vector interrupt occurs with an address is written to the VIRPT register. The DSP clears VIPD on return from the VIRPT interrupt service routine. Systems using vector interrupts should monitor VIPD to determine that the DSP has serviced the VIRPT interrupt and is ready for another vector interrupt.
15-14	HPS	**Host Packing Status.** These bits indicate the host's packing status as 00=pack complete (reset value), 01=1st stage pack/unpack, 10=2nd stage multi-stage pack/unpack, 11=reserved. These bits are read-only. The DSP clears these bits when DEN is cleared (changes from 1 to 0).
19-16	CRAT	**Core Clock-to-CLKIN ratio.** These bits indicate the state of the CLK_CFG3-0 pins (clock ratio) on the DSP. The reset value of CRAT is undefined.
31-20		Reserved (reset value =0)

External Port DMA Buffer Registers (EPBx)

These registers' addresses are `EPB0`—`0x04`, `EPB1`—`0x06`, `EPB2`—`0x14`, and `EPB3`—`0x16`. The reset value for these registers is undefined.

External port buffers are 8 levels deep and 64 bits wide. The buffers contain 40/48- or 32/64-bit words, depending on the external port buffer's data type selected with the `DTYPE` bit in the port's `DMACx` register. If the buffer contains 32-, 40- or 48-bit words, the port aligns the data with the lower bits of the buffer and zero fills the upper 32, 24 or 16 bits.

Normally, a DMA process automatically accesses the buffer register for memory transfer. Programs can also access these buffers as registers, but to access the full width of the buffer programs must use the `PX` register. A `PX` register move can access the entire 64 bits of an external port buffer using the full width `PX`.

Message Registers (MSGRx)

These registers' addresses are MSGR0—0x08, MSGR1—0x09, MSGR2—0x0a, MSGR3—0x0b, MSGR4—0x0c, MSGR5—0x0d, MSGR6—0x0e, and MSGR7—0x0f. The reset value for these registers is undefined.

PC Shadow Register (PC_SHDW)

This register's address is 0x10. The reset value for this register matches the PC register. PC_SHDW contains a read-only mirror of the 24-bit address in the Program Counter (PC) register. External devices can poll this PC_SHDW for the contents of PC. Note that the value in PC_SHDW may lag behind the current PC by one or more core clock cycles.

MODE2 Shadow Register (MODE2_SHDW)

This register's address is 0x11 (Figure A-2). In silicon revisions prior to revision 1.2, the upper 7 bits of the MODE2_SHDW register were documented to mirror corresponding bits within the MODE2 register. In silicon revision 1.2 (and any later revisions which may follow), MODE2_SHDW has been modified to no longer mirror the upper 7 bits of MODE2 and will contain information unique to the silicon revision. The reset value varies with the system setting and silicon revision. External devices can poll this MODE2_SHDW for the DSP's processor ID and silicon revision.

Bus Timeout Maximum Register (BMAX)

This register's address is 0x18. The reset value for this register is 0x0000 0000. The lower 16 bits of this register hold the value for the maximum number of cycles (-2) that the DSP can retain bus mastership. The upper 16 bits of this register are reserved.

For more information describing how BMAX and BCNT work, see "Mastership Timeout Bus" on page -108.

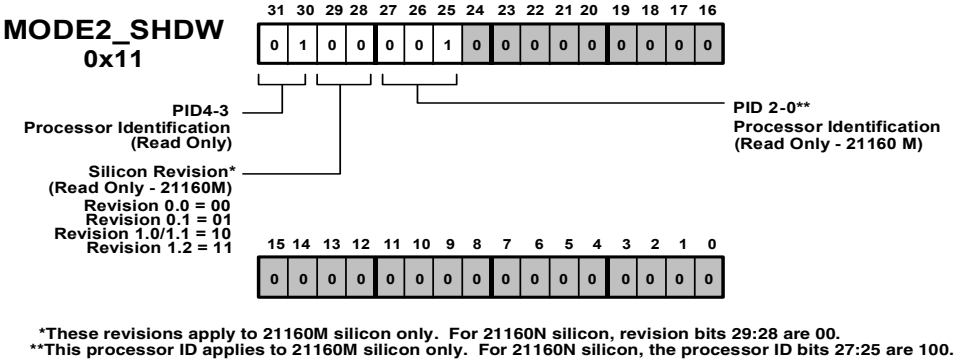

MODE2_SHDW
0x11
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
PID4-3
Processor Identification
(Read Only)
Silicon Revision*
(Read Only - 21160M)
Revision 0.0 = 00
Revision 0.1 = 01
Revision 1.0/1.1 = 10
Revision 1.2 = 11
PID 2-0**
Processor Identification
(Read Only - 21160 M)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*These revisions apply to 21160M silicon only. For 21160N silicon, revision bits 29:28 are 00.
**This processor ID applies to 21160M silicon only. For 21160N silicon, the processor ID bits 27:25 are 100.

Figure A-2. MODE2 Shadow Register

Bus (Timeout) Counter Register (BCNT)

This register's address is 0x19. The reset value for this register is 0x0000 0000. The lower 16 bits of this register hold the count of the number of cycles remaining for the DSP to retain bus mastership. The upper 16 bits of this register are reserved.

For more information describing how BMAX and BCNT work, see "Mastership Timeout Bus" on page -108.

Address of Last DRAM Page Register (ELAST)

This register's address is 0x1b. The reset value for this register is 0x0000 0000. For information describing how ELAST works, see "DRAM Page Boundary Detection" on page -15.

External Port DMA Control Registers (DMACx)

These registers' addresses are DMAC10–0x1c, DMAC11–0x1d, DMAC12–0x1e, and DMAC13–0x1f. The reset value for these registers is 0x0000 0000 unless you are booting from a host processor or PROM booting.

Each external port DMA channel has its own control register. The registers are DMAC10, DMAC11, DMAC12, and DMAC13, and they corresponding to DMA channels 10, 11, 12, and 13.

Table A-21 shows the contents of the DMACx registers.

Except for the FLSH bit, the control bits in the DMACx registers have a one-cycle effect latency (take effect the second cycle after being changed). The FLSH bit has a two-cycle effect latency.

Table A-21. External Port DMA Control Registers (DMACx) Bit Definitions

Bit(s)	Name	Definition
0	DEN	**External Port DMA Enable.** This bit enables (if set, =1) or disables (if cleared, =0) DMA for the corresponding external port FIFO buffer (EPBx).
1	CHEN	**External Port DMA Chaining Enable.** This bit enables (if set, =1) or disables (if cleared, =0) DMA chaining for the corresponding external port FIFO buffer (EPBx).
2	TRAN	**External Port Transmit/Receive Select.** This bit selects the transfer direction (transmit if set, =1) (receive if cleared, =0) for the corresponding external port FIFO buffer (EPBx).
4-3	PS	**External Port Packing Status.** These bits indicate the corresponding FIFO buffer's packing status as 00=pack complete, 01=1st stage pack/unpack, 10=2nd stage multi-stage pack/unpack, 11=reserved. These bits are read-only. The DSP clears these bits when DEN is cleared (changes from 1 to 0).
5	DTYPE	**External Port Data Type Select.** This bit selects the transfer data type (40/48=bit, 3-column if set, =1) (32/64-bit, 4-column if cleared, =0) for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's DTYPE setting while the buffer is enabled. The buffer's DTYPE setting overrides the internal memory block's setting IMDWx for Normal word width. Whether buffer is set for 48- or 64- bit words, programs must index (IIx) the corresponding DMA channel with a Normal word address; always an even address 64-bit.

Table A-21. External Port DMA Control Registers (DMACx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
8-6	PMODE	**External Port Packing Mode.** These bits select the packing mode for the corresponding external port FIFO buffer (EPBx) as follows: 000=No pack, 001=16 external to 32/64 internal packing, 010=16 external to 48 internal packing, 011=32 external to 48 internal packing, 100=32 external to 32/64 internal packing, 101=110=111=reserved. Programs must not change a buffer's PMODE setting while the buffer is enabled. For host processor accesses through the external port, the buffer's PMODE setting must match the Host Packing Mode (HPM) setting in the SYSCON registers.
9	MSWF	**Most Significant 16-bit Word First during packing.** When the buffer's PMODE is 001 or 010, this bit selects the packing order of 16-bit words (most significant first set, =1) (least significant first cleared, =0) for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's MSWF setting while the buffer is enabled.
10	MASTER	**Master Mode Enable.** This bit enables (if set, =1) or disables (if cleared, =0) master mode for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's MASTER setting while the buffer is enabled. The MASTER, HSHAKE, and EXTERN bits work together to select the external port buffer's mode. For more information, see Table 6-4 on page 6-22.
11	HSHAKE	**Handshake Mode Enable.** This bit enables (if set, =1) or disables (if cleared, =0) handshake mode for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's HSHAKE setting while the buffer is enabled. The MASTER, HSHAKE, and EXTERN bits work together to select the external port buffer's mode. For more information, see Table 6-4 on page 6-22.
12	INTIO	**Single-Word Interrupt Enable.** This bit enables (if set, =1) or disables (if cleared, =0) single-word, non-DMA, interrupt-driven transfers for the corresponding external port FIFO buffer (EPBx). To avoid spurious interrupts, programs must not change a buffer's INTIO setting while the buffer is enabled.

Table A-21. External Port DMA Control Registers (DMACx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
13	EXTERN	**External Handshake Mode Enable.** This bit enables (if set, =1) or disables (if cleared, =0) external handshake mode for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's EXTERN setting while the buffer is enabled. The MASTER, HSHAKE, and EXTERN bits work together to select the external port buffer's mode. For more information, see Table 6-4 on page 6-22.
14	FLSH	**Flush DMA Buffers and Status.** This bit flushes (when set, =1) settings for the corresponding external port FIFO buffer (EPBx). Flushing these settings does the following: • Clears (=0) the FS and PS status bits • Clears (=0) the FIFO buffer and DMA request counter • Clears (-0) any partially packed words When a program sets (=1) FLSH, the DSP flushes the settings and clears (=0) FLSH. There is a two-cycle effect latency in completing the flush operation. Programs must not set a buffer's FLSH during the same write that enables the buffer. Also, programs must not set a buffer's FLSH bit while the DMA channel is active. Programs should determine the channel's active status by reading the corresponding bit in the DMASTAT register.
15	PRIO	**External Port Bus Priority.** This bit selects the external bus access priority level (high if set, =1) (low if cleared, =0) for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's PRIO setting while the buffer is enabled. When PRIO is set, the DSP asserts the $\overline{\text{PA}}$ pin as part of external bus arbitration for DMA accesses using this buffer. The PRIO bit does not effect internal DMA priority arbitration.

Table A-21. External Port DMA Control Registers (DMACx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
17-16	FS	**External Port FIFO Buffer Status.** These bits indicate the corresponding FIFO buffer's status as 00=buffer empty, 01=buffer-not-full, 10=buffer-not-empty, 11=buffer full. For transmit (TRAN=1), buffer-not-full means that the buffer has space for one Normal word, and buffer-not-empty means that the buffer has space for two-or-more Normal words. For receive (TRAN=0), buffer-not-full means that the buffer contains one Normal word, and buffer-not-empty means that the buffer contains two-or-more Normal words. Any type of full status (01, 10, or 11) in receive mode indicates that new (unread) data is in the buffer. These bits are read-only. The DSP clears these bits when DEN is cleared (changes from 1 to 0).
18	INT32	**Internal Memory 32-bit Transfers Select.** This bit selects the external bus access width (32-bit transfers only if set, =1) (64-bit transfers when possible if cleared, =0) for the corresponding external port FIFO buffer (EPBx). Programs must not change a buffer's INT32 setting while the buffer is enabled. Note that the buffer's DTYPE and internal memory block's IMDWx setting (either can select 40/48-bit transfers) overrides a 32-bit transfers only (INT32 =1) setting.
20-19	MAXBL	**Maximum Burst Length Select.** These bits select the maximum burst transfer length for the corresponding external port FIFO buffer (EPBx) as follows: 00=burst disabled, 01=burst limit of 4, 10=11=reserved DSPs may perform burst accesses to external memory banks only when the bank is configured for synchronous access (EBxAM field in WAIT register). For burst writes, the memory bank's EBxAM must be configured for the one-wait state write, synchronous access mode. Burst reads are available in either the one- or two-waitstate write, synchronous access modes.
21-31		reserved

Internal Memory DMA Index Registers (IIx)

These registers' addresses are `II4—0x30`, `II5—0x38`, `II10—0x40`, `II11—0x48`, `II12—0x50`, `II13—0x58`, `II0—0x60`, `II1—0x68`, `II2—0x70`, `II3—0x78`, `II6—`

0x80, II7–0x88, II8–0x90, and II9–0x98. The reset value for these registers is undefined. An IIx register holds an address and acts as a pointer to memory for a DMA transfer.

For more information, see “I/O Processor” on page -1.

Internal Memory DMA Modifier Registers (IMx)

These registers’ addresses are IM4–0x31, IM5–0x39, IM10–0x41, IM11–0x49, IM12–0x51, IM13–0x59, IM0–0x61, IM1–0x69, IM2–0x71, IM3–0x79, IM6–0x81, IM7–0x89, IM8–0x91, and IM9–0x99. The reset value for these registers is undefined. An IMx register provides the increment or step size by which an IIx register is post-modified during a DMA operation.

For more information, see “I/O Processor” in Chapter , .

Internal Memory DMA Count Registers (Cx)

These registers’ addresses are C4–0x32, C5–0x3a, C10–0x42, C11–0x4a, C12–0x52, C13–0x5a, C0–0x62, C1–0x6a, C2–0x72, C3–0x7a, C6–0x82, C7–0x8a, C8–0x92, and C9–0x9a. The reset value for these registers is undefined. A Cx register holds the word count for a DMA transfer.

For more information, see “I/O Processor” on page -1.

Chain Pointer For Next DMA TCB Registers (CPx)

These registers’ addresses are CP4–0x33, CP5–0x3b, CP10–0x43, CP11–0x4b, CP12–0x53, CP13–0x5b, CP0–0x63, CP1–0x6b, CP2–0x73, CP3–0x7b, CP6–0x83, CP7–0x8b, CP8–0x93, and CP9–0x9b. The reset value for these registers is undefined. A CPx register holds the address for the next transfer control block in a chained DMA operation.

For more information, see “I/O Processor” on page -1.

General Purpose DMA Registers (GPx, DBx, DAx)

These registers' addresses are:

- GP4—0x34, GP5—0x3c, GP10—0x44, GP11—0x4c, GP12—0x54, GP13—0x5c, GP0—0x64, GP1—0x6c, GP2—0x74, GP3—0x7c, GP6—0x84, GP7—0x8c, GP8—0x94, and GP9—0x9c. The reset value for these registers is undefined.
- DB4—0x35, DB5—0x3d, DB0—0x65, DB1—0x6d, DB2—0x75, DB3—0x7d, DB6—0x85, DB7—0x8d, DB8—0x95, and DB9—0x9d. The reset value for these registers is undefined.
- DA4—0x36, DA5—0x3e, DA0—0x66, DA1—0x6e, DA2—0x76, DA3—0x7e, DA6—0x86, DA7—0x8e, DA8—0x96, and DA9—0x9e. The reset value for these registers is undefined.

In single-dimensional DMA operations, these registers are general purpose, but in two-dimensional DMA operations these registers have specific roles. These roles are:

- The channel's GPx register holds the 2-D DMA's Y count.
- The channel's DBx register holds the 2-D DMA's Y modifier.
- The channel's DAx register holds a chained 2-D DMA's initial.

Only Link Port and Serial Port DMA channels have DBx and DAx registers, because only these channels support 2-D DMA.

DMA Channel Status Register (DMASTAT)

This register's address is 0x37. The reset value for this register is undefined.

The lower bits in the DMASTAT register indicate DMA channel activity. Bits 0 through 13 correspond to channels 0 through 13 and indicate DMA status for each channel as active (if set, =1) or inactive (if cleared, =0). The

upper bits in the DMASTAT register indicate DMA chaining status. Bits 16 through 29 correspond to channels 0 through 13 and indicate DMA chaining status for each channel as enabled/pending (if set, =1) or disabled (if cleared, =0).

If a system needs full I/O bandwidth, do not frequently poll DMASTAT with the core. Using interrupts and/or polling the peripheral's individual buffer provides additional information on the status of a DMA transfer without having to poll the DMASTAT register.

Note that there is a single cycle of read latency between a change in a DMA channel's status and the update of its DMASTAT bit(s).

External Memory DMA Index Registers (EIx)

These registers' addresses are EI10—0x45, EI11—0x4d, EI12—0x55, and EI13—0x5d. The reset value for these registers is undefined. An EIx register holds an external memory address and acts as a pointer to memory for an external port DMA transfer.

For more information, see "I/O Processor" on page -1.

Only External Port DMA channels have EIx registers, because only these channels address DSP external memory.

External Memory DMA Modifier Registers (EMx)

These registers' addresses are EM10—0x46, EM11—0x4e, EM12—0x56, and EM13—0x5e. The reset value for these registers is undefined. An EMx register provides the increment or step size by which an EIx register is post-modified during an external port DMA operation. For more information, see "Overview" in Chapter , .

Only External Port DMA channels have EMx registers, because only these channels address DSP external memory.

External Memory DMA Count Registers (ECx)

These registers' addresses are: EC10—0x47, EC11—0x4f, EC12—0x57, and EC13—0x5f. The reset value for these registers is undefined. An ECx register holds the word count for an external port DMA transfer. For more information, see "Overview" in Chapter , .

Only External Port DMA channels have ECx registers, because only these channels address DSP external memory.

Link Port Buffer Registers (LBUFx)

These registers' addresses are LBUF0—0xc0, LBUF1—0xc2, LBUF2—0xc4, LBUF3—0xc6, LBUF4—0xc8, and LBUF5—0xca. The reset value for these registers is undefined.

Link port buffers are two levels deep and 48 bits wide. The buffers contain 32- or 48-bit words, depending on the link port's extended word size selected with the LxEXT bit in the port's LCTLx register. If the buffer contains 32-bit words, the port aligns the data with the lower 32 bits of the buffer and zero fills the upper 16 bits.

Normally, a DMA process automatically accesses the buffer register for memory transfer. Programs can also access these buffers as registers, but to access the full width of the buffer programs must use the PX register. A PX register move can access the entire 48 bits of a link buffer using the lower 48 bits of PX.

Link Port Buffer Control Registers (LCTLx)

These registers' addresses are LCTL0—0xcc and LCTL1—0xcd. The reset value for these registers is 0x0000 0000. Table A-22 and Table A-23 describe the bit fields within these registers. To avoid spurious interrupts, programs should mask Link Service Requests (LSRQ) before modifying the LCTLx registers.

For more information, see “Link Port Service Request and Mask Register (LSRQ)” on page A-68.

Table A-22. Link Port Buffer Control Registers (LCTLx) Bit Fields

Register	Bit(s)	Definition
LCTL0	0-9	Link Buffer 0 controls
	10-19	Link Buffer 1 controls
	20-29	Link Buffer 2 controls
	30-31	reserved
LCTL1	0-9	Link Buffer 3 controls
	10-19	Link Buffer 4 controls
	20-29	Link Buffer 5 controls
	30-31	reserved
Definitions appear in Table A-23 for each Link Buffer’s control bits (where x=0,1,2,3,4,5)		

Table A-23. Link Port Buffer Control Registers (LCTLx) Bit Definitions

Bit(s)	Name	Definition
0	LxEN	**Link Buffer Enable.** This bit enables (if set, =1) or disables (if cleared, =0) the corresponding link buffer (LBUFx). When the DSP disables the buffer (LxEN transitions from high to low), the DSP clears the corresponding LxSTAT and LxRERR bits.
1	LxDEN	**Link Buffer DMA Enable.** This bit enables (if set, =1) or disables (if cleared, =0) DMA transfers for the corresponding link buffer (LBUFx).
2	LxCHEN	**Link Buffer DMA Chaining Enable.** This bit enables (if set, =1) or disables (if cleared, =0) DMA chaining for the corresponding link buffer (LBUFx).
3	LxTRAN	**Link Buffer Transfer Direction.** This bit selects the transfer direction (transmit if set, =1) (receive if cleared, =0) for the corresponding link buffer (LBUFx).
Note that the Bit(s) column lists the bit’s position in group for a link buffer.		

Table A-23. Link Port Buffer Control Registers (LCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
4	LxEXT	**Link Buffer Extended Word Size.** This bit selects the transfer extended word size (48-bit if set, =1) (32-bit if cleared, =0) for the corresponding link buffer (LBUFx). Programs must not change a buffer's LxEXT setting while the buffer is enabled. The buffer's LxEXT setting overrides the internal memory block's setting IMDWx for Normal word width. Whether buffer is set for 48- or 32- bit words, programs must index (IIx) the corresponding DMA channel with a Normal word address.
6-5	LxCLKD	**Link Port Clock Divisor.** These bits select the transfer clock divisor for the corresponding link buffer (LBUFx). The transfer clock equals the processor core clock divided by LxCLKD, where LxCLKD[6-5] is: 01=1, 10=2, 11=3, or 00=4.
7	LxDMA2D	**Link Buffer DMA 2-Dimensional.** This bit enables (if set, =1) or disables (if cleared, =0) two-dimensional DMA transfers for the corresponding link buffer (LBUFx).
8	LxPDRDE	**Link Port pull-down Resistor Disable Enable.** This bit disables (if set, =1) or enables (if cleared, =0) the internal pull-down resistors on the LxCLK, LxACK, and LxDAT7-0 pins of the corresponding unassigned or disabled link port; this bit applies to the port, which is not necessarily the port assigned to the corresponding link buffer (LBUFx). Systems should not leave link port pins (LxCLK, LxACK, and LxDAT7-0) unconnected without clearing the corresponding LxPDRDE bit (enables 50KΩ internal pull-down resistors) or applying an external pull-down. In systems where several DSPs share a link port, only one DSP should have this bit cleared.
Note that the Bit(s) column lists the bit's position in group for a link buffer.		

Table A-23. Link Port Buffer Control Registers (LCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
9	LxDPWID	**Link Port Data Path Width.** This bit selects the link port data path width (8-bit if set, =1) (4-bit if cleared, =0) for the corresponding link port; this bit applies to the port, which is not necessarily the port assigned to the corresponding link buffer (LBUFx). Systems using a 4-bit should connect the lower link port data pins (LxDAT3-0) for data transfers and leave the upper pins (LxDAT7-4) unconnected. In 4-bit mode, the DSP applies pull-downs to the upper pins.
Note that the Bit(s) column lists the bit's position in group for a link buffer.		

Link Port Common Control Register (LCOM)

This register's address is `0xce`. The reset value for this register is `0x0000 0000` (see Table A-24).

Table A-24. Link Common Control Register (LCOM) Bit Definitions

Bit(s)	Name	Definition
1-0	L0STAT(1-0)	**Link Buffer 0 Status.** These bits indicate the corresponding buffer's status as 11=full, 00=empty, 01=reserved, 10=one word. When transmitting, these bits indicate when the buffer has space for more data. When receiving, these status bits indicate when the buffer contains new (unread) data. These bits are read-only. The DSP clears these bits when LxEN is cleared (changes from 1 to 0).
3-2	L1STAT(1-0)	**Link Buffer 1 Status.** (see L0STAT definition)
5-4	L2STAT(1-0)	**Link Buffer 2 Status.** (see L0STAT definition)
7-6	L3STAT(1-0)	**Link Buffer 3 Status.** (see L0STAT definition)
9-8	L4STAT(1-0)	**Link Buffer 4 Status.** (see L0STAT definition)
11-10	L5STAT(1-0)	**Link Buffer 5 Status.** (see L0STAT definition)

Table A-24. Link Common Control Register (LCOM) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
19-12	reserved	reserved
20	LMSP	**Link Buffer Mesh Multiprocessing.** This bit enables (if set, =1) or disable (if cleared, =0) mesh multiprocessing.
22-21	LPATHD	**Link Path (Mesh Multiprocessing) Delay.** These bits apply change over delays when changing to the next LPATH register as follows: 00=no additional delay, 01=1 additional delay, 10=2 additional delays, 11=3 additional delays. In a mesh multiprocessing applications, the change over delay can let the current receive operation complete on the current link port before the DSP selects a new link port. In some mesh multiprocessing applications, the changeover delay is significant.
25-23	reserved	reserved
26	LRERR0	**Link Buffer 0 Receive Pack Error Status.** These bits indicate the buffer's packing status as: 1=incomplete, 0=complete. When the buffer is ready to receive and pack a a new word, the DSP clears (=0) LRERRx. If this bit remains set (=1) after the buffer receives a word, a link transfer error (for example, a clock glitch) has occurred. The DSP also clears the LRERRx bits when LxEN is cleared (changes from 1 to 0).
27	LRERR1	**Link Buffer 1 Receive Pack Error Status.** (see LRERR0 definition)
28	LRERR2	**Link Buffer 2 Receive Pack Error Status.** (see LRERR0 definition)
29	LRERR3	**Link Buffer 3 Receive Pack Error Status.** (see LRERR0 definition)
30	LRERR4	**Link Buffer 4 Receive Pack Error Status.** (see LRERR0 definition)
31	LRERR5	**Link Buffer 5 Receive Pack Error Status.** (see LRERR0 definition)

Link Port Assignment Register (LAR)

This register's address is `0xcf`. The `LAR` register assigns link port buffers to link ports (Table A-25). The reset value for this register is `0x0002 C688`, assigning Link Port 0 to Link Buffer 0, Link Port 1 to Link Buffer 1, Link Port 2 to Link Buffer 2, Link Port 3 to Link Buffer 3, Link Port 4 to Link Buffer 4, and Link Port 5 to Link Buffer 5.

Table A-25. Link Port Assignment Register (LAR) Bit Definitions

Bit(s)	Name	Definition
2-0	A0LB	**Link port assignment for LBUF0.** These bits assign a link port to link buffer 0 (LBUF0) as follows: AxLB BitsLink Port # 000Link Port 0 001Link Port 1 010Link Port 2 011Link Port 3 100Link Port 4 101Link Port 5 110reserved 111inactive buffer
5-3	A1LB	**Link port assignment for LBUF1.** These bits assign a link port to link buffer 1 (LBUF1) as shown for "A0LB" on page A-67.
8-6	A2LB	**Link port assignment for LBUF2.** These bits assign a link port to link buffer 2 (LBUF2) as shown for "A0LB" on page A-67.
11-9	A3LB	**Link port assignment for LBUF3.** These bits assign a link port to link buffer 3 (LBUF3) as shown for "A0LB" on page A-67.
14-12	A4LB	**Link port assignment for LBUF4.** These bits assign a link port to link buffer 4 (LBUF4) as shown for "A0LB" on page A-67.
17-15	A5LB	**Link port assignment for LBUF5.** These bits assign a link port to link buffer 5 (LBUF5) as shown for "A0LB" on page A-67.
31-18	reserved	reserved

Link Port Service Request and Mask Register (LSRQ)

This register's address is 0xd0. The reset value for this register is 0x0000 0000.The LSRQ register contains transmit and receive mask and status bits for each link port. The mask bits in LSRQ mask (disable if set, =1) or unmask (enable if cleared, =0) the status bits in LSRQ register (see Table A-26).

The status bits indicate whether a disabled link port (DEN=0) has a pending service request to receive or transmit data. When an LSRQ receive request status bit is set (LxRRQ=1), another DSP has requested to send data by setting the link port's clock (LxCLK=1). When an LSRQ transmit request status bit is set (LxTRQ=1), another DSP has requested more data by setting the link port's acknowledge (LxACK=1).

Table A-26. Link Port Service Request Register (LSRQ) Bit Definitions

Bit(s)	Name	Definition
3-0		reserved
4	L0TM	**Link Port 0 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L0TRQ status bit.
5	L0RM	**Link Port 0 receive mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L0RRQ status bit.
6	L1TM	**Link Port 1 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L1TRQ status bit.
7	L1RM	**Link Port 1 receive mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L1RRQ status bit.
8	L2TM	**Link Port 2 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L2TRQ status bit.
9	L2RM	**Link Port 2 receive mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L2RRQ status bit.
10	L3TM	**Link Port 3 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L3TRQ status bit.

Table A-26. Link Port Service Request Register (LSRQ) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
11	L3RM	Link Port 3 receive mask. This bit unmasks (if set, =1) or masks (if cleared, =0) the L3RRQ status bit.
12	L4TM	**Link Port 4 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L4TRQ status bit.
13	L4RM	**Link Port 4 receive mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L4RRQ status bit.
14	L5TM	**Link Port 5 transmit mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L5TRQ status bit.
15	L5RM	**Link Port 5 receive mask.** This bit unmasks (if set, =1) or masks (if cleared, =0) the L5RRQ status bit.
19-16		reserved
20	L0TRQ	**Link Port 0 transmit request status (read-only).** If set (=1), this bit indicates that link port 0 is disabled, but L0ACK is set (indicating an external transmit request).
21	L0RRQ	**Link Port 0 receive request status (read-only).** If set (=1), this bit indicates that link port 0 is disabled, but L0CLK is set (indicating an external receive request).
22	L1TRQ	**Link Port 1 transmit request status (read-only).** If set (=1), this bit indicates that link port 1 is disabled, but L1ACK is set (indicating an external transmit request).
23	L1RRQ	**Link Port 1 receive request status (read-only).** If set (=1), this bit indicates that link port 1 is disabled, but L1CLK is set (indicating an external receive request).
24	L2TRQ	**Link Port 2 transmit request status (read-only).** If set (=1), this bit indicates that link port 2 is disabled, but L2ACK is set (indicating an external transmit request).
25	L2RRQ	**Link Port 2 receive request status (read-only).** If set (=1), this bit indicates that link port 2 is disabled, but L2CLK is set (indicating an external receive request).
26	L3TRQ	**Link Port 3 transmit request status (read-only).** If set (=1), this bit indicates that link port 3 is disabled, but L3ACK is set (indicating an external transmit request).

Table A-26. Link Port Service Request Register (LSRQ) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
27	L3RRQ	**Link Port 3 receive request status (read-only).** If set (=1), this bit indicates that link port 3 is disabled, but L3CLK is set (indicating an external receive request).
28	L4TRQ	**Link Port 4 transmit request status (read-only).** If set (=1), this bit indicates that link port 4 is disabled, but L4ACK is set (indicating an external transmit request).
29	L4RRQ	**Link Port 4 receive request status (read-only).** If set (=1), this bit indicates that link port 4 is disabled, but L4CLK is set (indicating an external receive request).
30	L5TRQ	**Link Port 5 transmit request status (read-only).** If set (=1), this bit indicates that link port 5 is disabled, but L5ACK is set (indicating an external receive request).
31	L5RRQ	**Link Port 5 receive request status (read-only).** If set (=1), this bit indicates that link port 5 is disabled, but L5CLK is set (indicating an external receive request).

Link Port Path Registers (LPATHx)

These registers' addresses are `LPATH1`—`0xd1`, `LPATH2`—`0xd2`, and `LPATH3`—`0xd3`. The reset value for these registers is undefined. The `LPATHx` registers only apply to mesh multiprocessing systems.

Link Port Path Counter Register (LPCNT)

This register's address is `0xd4`. The reset value for this register is undefined. The `LPCNT` register only applies to mesh multiprocessing systems.

Link Port Constant Registers (CNSTx)

These registers' addresses are `CNST1`—`0xd5` and `CNST2`—`0xd6`. The reset value for these registers is undefined. The `CNSTx` registers only apply to mesh multiprocessing systems.

SPORT Serial Transmit Control Registers (STCTLx)

These registers' addresses are `STCTL0`—`0xe0` and `STCTL1`—`0xf0`. The reset value for these registers is `0x0000 0000`. `STCTL0` is the transmit control register for serial port 0 (SPORT0), and `STCTL1` is the transmit control register for serial port 1 (SPORT1) (see Table A-27).

When changing SPORT operating modes, programs should clear a serial port's control register before writing new settings to the control register.

Table A-27. Serial Port Transmit Control Registers (STCTLx) Bit Definitions

Bit(s)	Name	Definition
0	SPEN[1]	**Serial Port Enable.** This bit enables (if set, =1) or disables (if cleared, =0) the corresponding serial port.
2-1	DTYPE	**Data Type Select.** These bits select the data type formatting for normal and multi-channel transmissions as follows: NormalMultiData Type Formatting 00x0Right-justify, zero-fill unused MSBs 01x1Right-justify, sign-extend unused MSBs 100xCompand using μ-law 111xCompand using A-law
3	SENDN	**Serial Word Endian Select.** This bit selects little endian words (LSB first, if set, =1) or big endian words (MSB first, if cleared, =0).
8-4	SLEN	**Serial Word Length Select (–1).** These bits select the word length (–1) in bits. Word sizes can be from 3-bit (SLEN=2) to 32-bit (SLEN=31).

Table A-27. Serial Port Transmit Control Registers (STCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
9	PACK	**16-bit to 32-bit Word Packing Enable.** This bit enables (if set, =1) or disables (if cleared, =0) 16- to 32-bit word packing.
10	ICLK[1]	**Internal Transmit Clock Select.** This bit selects the internal transmit clock (if set, =1) or external transmit clock (if cleared, =0).
11		reserved
12	CKRE	**Clock Rising Edge Select.** This bit select whether the serial port uses the rising edge (if set, =1) or falling edge (if cleared, =0) of the clock signal for sampling data and the frame sync.
13	TFSR[1]	**Transmit Frame Sync Required Select.** This bit selects whether the serial port requires (if set, =1) or does not require (if cleared, =0) a transfer frame synch.
14	ITFS[1]	**Internally Transmit Frame Sync Select.** This bit selects whether the serial port uses an internal TFS (if set, =1) or uses an external TFS (if cleared, =0).
15	DITFS	**Data Independent Transmit Frame Sync Select.** This bit selects whether the serial port uses a data-independent TFS (synch at selected interval, if set, =1) or uses a data-dependent TFS (synch when data in TX, if cleared, =0).
16	LTFS	**Low Active Transmit Frame Synch Select.** This bit selects an active low TFS (if set, =1) or active high TFS (if cleared, =0).
17	LAFS[1]	**Late Transmit Frame Sync Select.** This bit selects a late TFS (TFS during first bit, if set, =1) or an early TFS (TFS before first bit, if cleared, =0).
18	SDEN	**Serial Port DMA Enable.** This bit enables (if set, =1) or disables (if cleared, =0) the serial port's DMA.
19	SCHEN	**Serial Port DMA Chaining Enable.** This bit enables (if set, =1) or disables (if cleared, =0) serial port DMA chaining.
23-20	MFD	**Multichannel Transmit Frame Sync Delay Select.** These bits select the delay in serial clock cycles between the TFS and the first data bit. When MFD=0, the TFS and first data bit are concurrent. The maximum value is MFD=16.

Table A-27. Serial Port Transmit Control Registers (STCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
28-24	CHNL[2]	**Current Channel Selected (read-only).** These bits indicate which channel the DSP has selected for the serial port's transmission in multichannel mode.
29	TUVF[2]	**Transmit Underflow Status (sticky, read-only).** This bit indicates whether the serial transmit operation has underflowed (if set, =1).
31-30	TXS[2]	**Transmit Data Buffer Status (read-only).** These bits indicate the status of the serial port's transmit buffer (TXx) as follows: 11=full, 00=empty, 10=partially full.

1 This bit must be cleared for multichannel operation.
2 Status bits are read-only. Disabling the serial port (setting SPEN=0), clears the status bits. TXS may change state if the data is read or written by the processor core while the SPORT is disabled.

SPORT Serial Receive Control Registers (SRCTLx)

These registers' addresses are `SRCTL0`—`0xe1` and `SRCTL1`—`0xf1`. The reset value for these registers is `0x0000 0000`. `SRCTL0` is the receive control register for serial port 0 (SPORT0), and `SRCTL1` is the receive control register for serial port 1 (SPORT1) (see Table A-28).



When changing SPORT operating modes, programs should clear a serial port's control register before writing new settings to the control register.

SPORT Transmit Buffer Registers (TXx)

These registers' addresses are `TX0`—`0xe2` and `TX1`—`0xf2`. The reset value for these registers is undefined. The 32-bit `TXx` registers hold the output data for serial port transmit operations. For more information on how transmit buffers work, see "Transmit and Receive Data Buffers (TX, RX)" on page -12.

Table A-28. Serial Port Receive Control Registers (SRCTLx) Bit Definitions

Bit(s)	Name	Definition
0	SPEN[1]	**Serial Port Enable.** This bit enables (if set, =1) or disables (if cleared, =0) the corresponding serial port.
2-1	DTYPE	**Data Type Select.** These bits select the data type formatting for normal and multi-channel reception as follows: NormalMultiData Type Formatting 00x0Right-justify, zero-fill unused MSBs 01x1Right-justify, sign-extend unused MSBs 100xCompand using μ-law 111xCompand using A-law
3	SENDN	**Serial Word Endian Select.** This bit selects little endian words (LSB first, if set, =1) or big endian words (MSB first, if cleared, =0).
8-4	SLEN	**Serial Word Length Select (–1).** These bits select the word length (–1) in bits. Word sizes can be from 3-bit (SLEN=2) to 32-bit (SLEN=31).
9	PACK	**16-bit to 32-bit Word Packing Enable.** This bit enables (if set, =1) or disables (if cleared, =0) 16- to 32-bit word packing.
10	ICLK	**Internally Receive Clock Select.** This bit selects the internal receive clock (if set, =1) or external receive clock (if cleared, =0).
11		reserved
12	CKRE	**Clock Rising Edge Select.** This bit select whether the serial port uses the rising edge (if set, =1) or falling edge (if cleared, =0) of the clock signal for sampling data and the frame sync.
13	RFSR[1]	**Receive Frame Sync Required Select.** This bit selects whether the serial port requires (if set, =1) or does not require (if cleared, =0) a receive frame synch.
14	IRFS[1]	**Internal Receive Frame Sync Select.** This bit selects whether the serial port uses an internal RFS (if set, =1) or uses an external RFS (if cleared, =0).

Table A-28. Serial Port Receive Control Registers (SRCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
15	IMODE (multichannel mode only)	**Receive Comparison Select.** This bit works in conjunction with the IMAT bit to select a receive comparison in multichannel mode. If IMAT and IMODE are set (=1), the DSP accepts receive data if the KEYWD comparison is true. If IMAT is cleared (=0) and IMODE is set (=1), the DSP accepts receive data if the KEYWD comparison is false. This bit is cleared in single channel mode.
16	LRFS	**Low Active Receive Frame Synch Select.** This bit selects an active low RFS (if set, =1) or active high RFS (if cleared, =0).
17	LAFS[1]	**Late Receive Frame Sync Select.** This bit selects a late RFS (RFS during first bit, if set, =1) or an early RFS (RFS before first bit, if cleared, =0).
18	SDEN	**Serial Port DMA Enable.** This bit enables (if set, =1) or disables (if cleared, =0) the serial port's DMA.
19	SCHEN	**Serial Port DMA Chaining Enable.** This bit enables (if set, =1) or disables (if cleared, =0) serial port DMA chaining.
20	IMAT (multichannel mode only)	This bit works in conjunction with the IMODE bit to select a receive comparison in multichannel mode. If IMAT and IMODE are set (=1), the DSP accepts receive data if the KEYWD comparison is true. If IMAT is cleared (=0) and IMODE is set (=1), the DSP accepts receive data if the KEYWD comparison is false. This bit is reserved in all other modes.
21	D2DMA[1]	**Two Dimensional DMA Array Enable.** This bit enables (if set, =1) or disables (if cleared, =0) two-dimensional serial DMA.
22	SPL[1]	**Serial Port Loopback Enable.** This bit enables (if set, =1) or disables (if cleared, =0) serial port loopback mode.
23	MCE	**Multichannel Enable.** This bit enables (if set, =1) or disables (if cleared, =0) multichannel serial port mode.
28-24	NCH	**Number of Multi Channels (–1) Select.** These bits select the number of channels (–1) for a multichannel serial port. The number of channels can be from 1 (NCH=0) to 32 (NCH=31).

Table A-28. Serial Port Receive Control Registers (SRCTLx) Bit Definitions (Cont'd)

Bit(s)	Name	Definition
29	ROVF[2]	**Receive Overflow Status (sticky, read-only).** This bit indicates whether the receive operation has overflowed (if set, =1).
31-30	RXS[2]	**Receive Data Buffer Status (read-only).** These bits indicate the status of the serial port's receive buffer (RXx) as follows: 11=full, 00=empty, 10=partially full.

1 This bit must be cleared for multichannel operation.
2 Status bits are read-only. Disabling the serial port (setting SPEN=0), clears the status bits. RXS may change state if the data is read or written by the processor core while the SPORT is disabled.

SPORT Receive Buffer Registers (RXx)

These registers' addresses are `RX0`—`0xe3` and `RX1`—`0xf3`. The reset value for these registers is undefined. The 32-bit `RXx` registers hold the input data from serial port receive operations. For more information on how transmit buffers work, see "Transmit and Receive Data Buffers (TX, RX)" on page -12.

SPORT Transmit Divisor Registers (TDIVx)

These registers' addresses are `TDIV0`—`0xe4` and `TDIV1`—`0xf4`. The reset value for these registers is undefined. These registers contain two fields:

Bits 15-0 are `TCLKDIV`. These bits select the Transmit Clock Divisor for internally generated `TCLK` as follows:

$$TCLKDIV = \frac{f_{CCLK}}{2(f_{TCLK})} - 1$$

Bits 31-16 are TFSDIV. These bits select the Transmit Frame Sync Divisor for internally generated TFS as follows:

$$TFSDIV = \frac{f_{TCLK}}{f_{TFS}} - 1$$

SPORT Transmit Count Registers (TCNTx)

These registers' addresses are TCNT0—0xe5 and TCNT1—0xf5. The reset value for these registers is undefined. The TCNTx registers only apply to mesh multiprocessing systems.

SPORT Receive Divisor Registers (RDIVx)

These registers' addresses are RDIV0—0xe6 and RDIV1—0xf6. The reset value for these registers is undefined. These registers contain two fields:

Bits 15-0 are RCLKDIV. These bits select the Receive Clock Divisor for internally generated RCLK as follows:

$$RCLKDIV = \frac{f_{CCLK}}{2(f_{RCLK})} - 1$$

Bits 31-16 are RFSDIV. These bits select the Receive Frame Sync Divisor for internally generated RFS as follows:

$$RFSDIV = \frac{f_{RCLK}}{f_{RFS}} - 1$$

SPORT Receive Count Registers (RCNTx)

These registers' addresses are RCNT0—0xe7 and RCNT1—0xf7. The reset value for these registers is undefined. The RCNTx registers only apply to mesh multiprocessing systems.

SPORT Transmit Select Registers (MTCSx)

These registers' addresses are MTCS0—0xe8 and MTCS1—0xf8. The reset value for these registers is undefined.

Each bit, 31-0, set (=1) in MTCSx, corresponds to an active transmit channel, 31-0, on a multichannel mode serial port. When the MTCSx register

activates a channel, the serial port transmits the word in that channel's position of the data stream. When a channel's bit in the MTCSx register is cleared (=0), the serial port's DT (data transmit) pin three-states during the channel's transmit time slot.

SPORT Receive Select Registers (MRCSx)

These registers' addresses are MRCS0—0xe9 and MRCS1—0xf9. The reset value for these registers is undefined.

Each bit, 31-0, set (=1) in MRCSx, corresponds to an active receive channel, 31-0, on a multichannel mode serial port. When the MRCSx register activates a channel, the serial port receives the word in that channel's position of the data stream and loads the word into the RXx buffer. When a channel's bit in the MRCSx register is cleared (=0), the serial port ignores any input during the channel's receive time slot.

SPORT Transmit Compand Registers (MTCCSx)

These registers' addresses are MTCCS0—0xea and MTCCS1—0xfa. The reset value for these registers is undefined.

Each bit, 31-0, set (=1) in MTCCSx, corresponds to an companded transmit channel, 31-0, on a multichannel mode serial port. When the MTCCSx register activates companding for a channel, the serial port applies the companding from the serial port's DTYPE selection to the transmitted word in that channel's position of the data stream. When a channel's bit in the MTCCSx register is cleared (=0), the serial port does not compand the output during the channel's receive time slot.

SPORT Receive Compand Register (MRCCSx)

These registers' addresses are MRCCS0—0xeb and MRCCS1—0xfb. The reset value for these registers is undefined.

Each bit, 31-0, set (=1) in MRCCSx, corresponds to an companded receive channel, 31-0, on a multichannel mode serial port. When the MRCCSx register activates companding for a channel, the serial port applies the companding from the serial port's DTYPE selection to the received word in that channel's position of the data stream. When a channel's bit in the MRCCSx register is cleared (=0), the serial port does not compand the input during the channel's receive time slot.

SPORT Receive Comparison and Mask Registers (KEYWDx and KEYMASKx)

These registers' addresses are KEYWD0—0xec, KEYMASK0—0xed, KEYWD1—0xfc, and KEYMASK1—0xfd. The reset value for these registers is undefined.

The receive comparison registers are 32-bit registers that aid multiprocessor communications when using multichannel mode (MCE=1) through the serial ports. The KEYWD0 or KEYWD1 register stores the pattern to be matched with the incoming data. The corresponding KEYMASK0 or KEYMASK1 register specifies which of the bits in the received data should be compared. Setting a KEYMASKx bit (=1) masks the corresponding bit in the KEYWDx register, disabling its comparison. For information on using receive comparison, see "SPORT Receive Comparison Registers" on page -33.

SPORT Serial Path Length Registers (SPATHx)

These registers' addresses are SPATH0—0xee and SPATH1—0xfe. The reset value for these registers is 0x0000 0001. The SPATHx registers only apply to mesh multiprocessing systems.



Note that for standard, non-mesh-multiprocessing operation of the serial ports, the SPATHx register must remain equal to the reset initialization value, 0x0001.

SPORT Serial Path Counter Registers (SPCNTx)

These registers' addresses are SPCNT0–0xef and SPCNT1–0xff. The reset value for these registers is 0x0000 0001. The SPCNT registers only apply to mesh multiprocessing systems.



Note that for standard, non-mesh-multiprocessing operation of the serial ports, the SPCNTx register must remain equal to the reset initialization value, 0x0001.

Register and Bit #Defines File (def21160.h)

The following example definitions file is for the ADSP-21160 DSP. For the most current definitions file, programs should use the version of this file that comes with the software development tools. The version of the file that appears here is included as a guide only.

```
/*****************************************************************
***********
 *
 * def21160.h
 *
 * (c) Copyright 2001 Analog Devices, Inc. All rights reserved.
 *

*****************************************************************
*********/

/*---------------------------------------------------------------
-----------
def21160.h - SYSTEM and IOP REGISTER BIT and ADDRESS DEFINITIONS
FOR ADSP-21160 DSP
```

```
This include file contains a list of macro "defines" to enable
the programmer
to use symbolic names for the following ADSP-21160 DSP
facilities:
 - instruction condition codes
 - system register bit definitions
 - IOP register map
 - *some* IOP register bit definitions

Here are some example uses:

   bit set mode1 BR0|IRPTEN|ALUSTAT;

   ustat1=BSO|HPM01|HMSWF;
   DM(SYSCON)=ustat1;

------------------------------------------------------------------*/
#ifndef __DEF21160_H_
#define __DEF21160_H_

/*-----------------------------------------------------------------*/
/*    System Register bit definitions                              */
/*-----------------------------------------------------------------*/
/* MODE1 and MMASK registers */
#define BR8    0x00000001 /* Bit 0: Bit-reverse for I8 */
#define BR0     0x00000002 /* Bit 1: Bit-reverse for I0 (uses
DMS0- only */
#defineSRCU0x00000004/* Bit 2: Alt. register select for comp.
units*/
#defineSRD1H0x00000008/* Bit 3: DAG1 alt. register select (7-4)
*/
#defineSRD1L0x00000010/* Bit 4: DAG1 alt. register select (3-0)
*/
```

```
#defineSRD2H0x00000020/* Bit 5: DAG2 alt. register select (15-12)
*/
#defineSRD2L0x00000040/* Bit 6: DAG2 alt. register select (11-8)
*/
#defineSRRFH0x00000080/* Bit 7: Register file alt. select for
R(15-8) */
#defineSRRFL0x00000400/* Bit 10: Register file alt. select for
R(7-0) */
#defineNESTM0x00000800/* Bit 11: Interrupt nesting enable*/
#defineIRPTEN0x00001000/* Bit 12: Global interrupt enable*/
#defineALUSAT0x00002000/* Bit 13: Enable ALU fixed-pt. saturation
*/
#defineSSE0x00004000/* Bit 14: Enable short word sign extension*/
#defineTRUNC0x00008000/* Bit 15: 1=fltg-pt. truncation 0=Rnd to
nearest */
#defineRND320x00010000/* Bit 16: 1=32-bit fltg-pt. rounding
0=40-bit rnd */
#defineCSEL0x00060000/* Bit 17-18: CSelect: Bus Mastership   */
#definePEYEN0x00200000/* Bit 21: Processing Element Y enable */
#defineSIMD0x00200000/* Bit 21: Enable SIMD Mode */
#defineBDCST90x00400000/* Bit 22: Load Broadcast for I9 */
#defineBDCST10x00800000/* Bit 23: Load Broadcast for I1 */
#defineCBUFEN0x01000000/* Bit 23: Circular Buffer Enable */

/* MODE2 register */
#defineIRQ0E0x00000001/* Bit 0: IRQ0- 1=edge sens. 0=level sens.
*/
#defineIRQ1E0x00000002/* Bit 1: IRQ1- 1=edge sens. 0=level sens.
*/
#defineIRQ2E0x00000004/* Bit 2: IRQ2- 1=edge sens. 0=level sens.
*/
#defineCADIS0x00000010/* Bit 4: Cache disable   */
#defineTIMEN0x00000020/* Bit 5: Timer enable    */
#defineBUSLK0x00000040/* Bit 6: External bus lock */
```

```
#defineFLG0O0x00008000/* Bit 15: FLAG0 1=output 0=input */
#defineFLG1O0x00010000/* Bit 16: FLAG1 1=output 0=input */
#defineFLG2O0x00020000/* Bit 17: FLAG2 1=output 0=input */
#defineFLG3O0x00040000/* Bit 18: FLAG3 1=output 0=input */
#defineCAFRZ0x00080000/* Bit 19: Cache freeze */
#defineIRAE0x00100000/* Bit 20: IOP Register Access Enable */
#defineU64MAE0x00200000/* Bit 21: Unaligned 64-bit Memory Access
Enable */
/* bits 31-30, 27-25 are Processor ID[4:0], read only, value:
0b01001
   bits 29-28    are silicon revision[1:0], read only, value: 0

   These bits (only) are routed to Mode2 Shadow register (IOP
register ox11)
*/

/* FLAGS register */
#define FLG0    0x00000001 /* Bit 0: FLAG0 value */
#define FLG1    0x00000002 /* Bit 1: FLAG1 value */
#define FLG2    0x00000004 /* Bit 2: FLAG2 value */
#define FLG3    0x00000008 /* Bit 3: FLAG3 value */

/* ASTATx and ASTATy registers */
#ifdef SUPPORT_DEPRECATED_USAGE
/* Several of these (AV, AC, MV, SV, SZ) are assembler-reserved
keywords,
   so this style is now deprecated. If these are defined, the
assembler-
   reserved keywords are still available in lowercase, e.g.,
           IF sz JUMP LABEL1.
*/
# define AZ   0x00000001 /* Bit 0: ALU result zero or fltg-pt.
underflow */
```

```
# define AV   0x00000002 /* Bit 1: ALU over-
flow                          */
# define AN   0x00000004 /* Bit 2: ALU result nega-
tive                    */
# define AC   0x00000008 /* Bit 3: ALU fixed-pt. carry
*/
# define AS   0x00000010 /* Bit 4: ALU X input sign (ABS and MANT
ops)   */
# define AI   0x00000020 /* Bit 5: ALU fltg-pt. invalid operation
*/
# define MN   0x00000040 /* Bit 6: Multiplier result nega-
tive             */
# define MV   0x00000080 /* Bit 7: Multiplier over-
flow                    */
# define MU   0x00000100 /* Bit 8: Multiplier fltg-pt. underflow
*/
# define MI   0x00000200 /* Bit 9: Multiplier fltg-pt. invalid
operation */
# define AF   0x00000400 /* Bit 10: ALU fltg-pt. opera-
tion                */
# define SV   0x00000800 /* Bit 11: Shifter over-
flow                       */
# define SZ   0x00001000 /* Bit 12: Shifter result zero
*/
# define SS   0x00002000 /* Bit 13: Shifter input sign
*/
# define BTF 0x00040000 /* Bit 18: Bit test flag for system reg-
isters    */
# define CACC0 0x01000000 /* Bit 24: Compare Accumulation Bit 0
*/
# define CACC1 0x02000000 /* Bit 25: Compare Accumulation Bit 1
*/
# define CACC2 0x04000000 /* Bit 26: Compare Accumulation Bit 2
*/
```

```
# define CACC3 0x08000000 /* Bit 27: Compare Accumulation Bit 3
*/
# define CACC4 0x10000000 /* Bit 28: Compare Accumulation Bit 4
*/
# define CACC5 0x20000000 /* Bit 29: Compare Accumulation Bit 5
*/
# define CACC6 0x40000000 /* Bit 30: Compare Accumulation Bit 6
*/
# define CACC7 0x80000000 /* Bit 31: Compare Accumulation Bit 7
*/

#endif

#define ASTAT_AZ 0x00000001 /* Bit 0: ALU result zero or fltg-pt.
u'flow */
#define ASTAT_AV 0x00000002 /* Bit 1: ALU over-
flow                        */
#define ASTAT_AN 0x00000004 /* Bit 2: ALU result nega-
tive                */
#define ASTAT_AC 0x00000008 /* Bit 3: ALU fixed-pt. carry
*/
#define ASTAT_AS 0x00000010 /* Bit 4: ALU X input signpost and
MANT ops)*/
#define ASTAT_AI 0x00000020 /* Bit 5: ALU fltg-pt. invalid opera-
tion    */
#define ASTAT_MN 0x00000040 /* Bit 6: Multiplier result negative
*/
#define ASTAT_MV 0x00000080 /* Bit 7: Multiplier over-
flow                */
#define ASTAT_MU 0x00000100 /* Bit 8: Multiplier fltg-pt. under-
flow     */
#define ASTAT_MI 0x00000200 /* Bit 9: Multiplier fltg-pt. invalid
pop.   */
```

```
#define ASTAT_AF 0x00000400 /* Bit 10: ALU fltg-pt. opera-
tion            */
#define ASTAT_SV 0x00000800 /* Bit 11: Shifter over-
flow                  */
#define ASTAT_SZ 0x00001000 /* Bit 12: Shifter result zero
*/
#define ASTAT_SS 0x00002000 /* Bit 13: Shifter input sign
*/
#define ASTAT_BTF 0x00040000 /* Bit 18: Bit test flag for system
registers*/
#define ASTAT_CACC0 0x01000000 /* Bit 24: Compare Accumulation
Bit 0      */
#define ASTAT_CACC1 0x02000000 /* Bit 25: Compare Accumulation
Bit 1      */
#define ASTAT_CACC2 0x04000000 /* Bit 26: Compare Accumulation
Bit 2      */
#define ASTAT_CACC3 0x08000000 /* Bit 27: Compare Accumulation
Bit 3      */
#define ASTAT_CACC4 0x10000000 /* Bit 28: Compare Accumulation
Bit 4      */
#define ASTAT_CACC5 0x20000000 /* Bit 29: Compare Accumulation
Bit 5      */
#define ASTAT_CACC6 0x40000000 /* Bit 30: Compare Accumulation
Bit 6      */
#define ASTAT_CACC7 0x80000000 /* Bit 31: Compare Accumulation
Bit 7      */

/* STKYx and STKYy registers */
#define AUS 0x00000001 /* Bit 0: ALU fltg-pt. under-
flow                */
#define AVS 0x00000002 /* Bit 1: ALU fltg-pt. over-
flow                 */
#define AOS 0x00000004 /* Bit 2: ALU fixed-pt. over-
flow                */
```

```
#define AIS 0x00000020 /* Bit 5: ALU fltg-pt. invalid opera-
tion       */
#define MOS 0x00000040 /* Bit 6: Multiplier fixed-pt. over-
flow        */
#define MVS 0x00000080 /* Bit 7: Multiplier fltg-pt. over-
flow         */
#define MUS 0x00000100 /* Bit 8: Multiplier fltg-pt. under-
flow        */
#define MIS 0x00000200 /* Bit 9: Multiplier fltg-pt. invalid
operation*/
#define CB7S 0x00020000 /* Bit 17: DAG1 circular buffer 7 over-
flow        */
#define CB15S 0x00040000 /* Bit 18: DAG2 circular buffer 15 over-
flow      */
#define PCFL 0x00200000 /* Bit 21: PC stack full
*/
#define PCEM 0x00400000 /* Bit 22: PC stack empty
*/
#define SSOV 0x00800000 /* Bit 23: Status stack overflow (MODE1
and ASTAT) */
#define SSEM 0x01000000 /* Bit 24: Status stack empty
*/
#define LSOV 0x02000000 /* Bit 25: Loop stack overflow
*/
#define LSEM 0x04000000 /* Bit 26: Loop stack empty
*/

/* STKYx register *ONLY* */
#define IRA 0x00080000 /* Bit 19: IOP Register Access
*/
#define U64MA 0x00100000 /* Bit 20: Unaligned 64-bit Memory
Access         */

/* IRPTL and IMASK and IMASKP registers */
```

```
#define EMUI    0x00000001 /* Bit 0: Offset: 00: Emulator Inter-
rupt       */
#define RSTI   0x00000002 /* Bit 1: Offset: 04: Reset
*/
#define IICDI 0x00000004 /* Bit 2: Offset: 08: Illegal Input Con-
dition
               Detected */
#define SOVFI 0x00000008 /* Bit 3: Offset: 0c: Stack over-
flow             */
#define TMZHI 0x00000010 /* Bit 4: Offset: 10: Timer = 0 (high
priority) */
#define VIRPTI 0x00000020 /* Bit 5: Offset: 14: Vector interrupt
*/
#define IRQ2I 0x00000040 /* Bit 6: Offset: 18: IRQ2- asserted
*/
#define IRQ1I 0x00000080 /* Bit 7: Offset: 1c: IRQ1- asserted
*/
#define IRQ0I 0x00000100 /* Bit 8: Offset: 20: IRQ0- asserted
*/
#define SPR0I 0x00000400 /* Bit 10: Offset: 28: SPORT0 receive
DMA channel */
#define SPR1I 0x00000800 /* Bit 11: Offset: 2c: SPORT1 receive
DMA channel */
#define SPT0I 0x00001000 /* Bit 12: Offset: 30: SPORT0 transmit
DMA channel */
#define SPT1I 0x00002000 /* Bit 13: Offset: 34: SPORT1 transmit
DMA channel */
#define LPISUMI 0x00004000 /* Bit 14: Offset: na: LPort Interrupt
Summary */
#define EP0I 0x00008000 /* Bit 15: Offset: 50: External port
channel 0 DMA */
#define EP1I 0x00010000 /* Bit 16: Offset: 54: External port
channel 1 DMA */
```

```
#define EP2I 0x00020000 /* Bit 17: Offset: 58: External port
channel 2 DMA */
#define EP3I 0x00040000 /* Bit 18: Offset: 5c: External port
channel 3 DMA */
#define LSRQI 0x00080000 /* Bit 19: Offset: 60: Link service
request     */
#define CB7I    0x00100000 /* Bit 20: Offset: 64: Circ. buffer 7
overflow   */
#define CB15I   0x00200000 /* Bit 21: Offset: 68: Circ. buffer 15
overflow */
#define TMZLI   0x00400000 /* Bit 22: Offset: 6c: Timer = 0 (low
priority) */
#define FIXI    0x00800000 /* Bit 23: Offset: 70: Fixed-pt.
overflow        */
#define FLTOI   0x01000000 /* Bit 24: Offset: 74: fltg-pt. over-
flow        */
#define FLTUI   0x02000000 /* Bit 25: Offset: 78: fltg-pt. under-
flow       */
#define FLTII   0x04000000 /* Bit 26: Offset: 7c: fltg-pt.
invalid         */
#define SFT0I   0x08000000 /* Bit 27: Offset: 80: user software
int 0      */
#define SFT1I   0x10000000 /* Bit 28: Offset: 84: user software
int 1      */
#define SFT2I   0x20000000 /* Bit 39: Offset: 88: user software
int 2      */
#define SFT3I   0x40000000 /* Bit 30: Offset: 8c: user software
int 3      */

/* LIRPTL register */
#define LP0I 0x00000001 /* Bit 0: Offset: 38: Link port channel 0
DMA */
#define LP1I 0x00000002 /* Bit 1: Offset: 3C: Link port channel 1
DMA */
```

```
#define LP2I 0x00000004 /* Bit 2: Offset: 40: Link port channel 2
DMA */
#define LP3I 0x00000008 /* Bit 3: Offset: 44: Link port channel 3
DMA */
#define LP4I 0x00000010 /* Bit 4: Offset: 48: Link port channel 4
DMA */
#define LP5I 0x00000020 /* Bit 5: Offset: 4C: Link port channel 5
DMA */
#define LP0MSK 0x00010000 /* Bit 16: Link port channel 0 Inter-
rupt Mask */
#define LP1MSK 0x00020000 /* Bit 17: Link port channel 1 Inter-
rupt Mask */
#define LP2MSK 0x00040000 /* Bit 18: Link port channel 2 Inter-
rupt Mask */
#define LP3MSK 0x00080000 /* Bit 19: Link port channel 3 Inter-
rupt Mask */
#define LP4MSK 0x00100000 /* Bit 20: Link port channel 4 Inter-
rupt Mask */
#define LP5MSK 0x00200000 /* Bit 21: Link port channel 5 Inter-
rupt Mask */
#define LP0MSKP 0x01000000 /* Bit 24: Link port channel 0 Inter-
rupt Mask
                           Pointer*/
#define LP1MSKP 0x02000000 /* Bit 25: Link port channel 1 Inter-
rupt Mask
                           Pointer */
#define LP2MSKP 0x04000000 /* Bit 26: Link port channel 2 Inter-
rupt Mask
                           Pointer */
#define LP3MSKP 0x08000000 /* Bit 27: Link port channel 3 Inter-
rupt Mask
                           Pointer */
```

```
#define LP4MSKP 0x10000000 /* Bit 28: Link port channel 4 Inter-
rupt Mask
                           Pointer */
#define LP5MSKP 0x20000000 /* Bit 29: Link port channel 5 Inter-
rupt Mask
                           Pointer */

/*-------------------------------------------------------------
---------*/
/*                I/O Processor Register Map
*/
/*-------------------------------------------------------------
----------*/
#define SYSCON 0x00     /* System configuration regis-
ter                */
#define VIRPT 0x01     /* Vector interrupt regis-
ter                    */
#define WAIT   0x02     /* External Port Wait register - renamed
to EPCON */
#define EPCON 0x02      /* External Port configuration regis-
ter            */
#define SYSTAT 0x03     /* System status regis-
ter                          */
/* the upper 32-bits of the 64-bit EPBxs are only accessible as
64-bit
reference */
#define EPB0   0x04    /* External port DMA buffer 0
*/
#define EPB1   0x06    /* External port DMA buffer 1
*/
#define MSGR0 0x08    /* Message register 0
*/
#define MSGR1 0x09    /* Message register 1
*/
```

```
#define MSGR2 0x0a    /* Message register 2
*/
#define MSGR3 0x0b    /* Message register 3
*/
#define MSGR4 0x0c    /* Message register 4
*/
#define MSGR5 0x0d    /* Message register 5
*/
#define MSGR6 0x0e    /* Message register 6
*/
#define MSGR7 0x0f    /* Message register 7
*/

/* IOP shadow registers of the core control regs
*/
#define PC_SHDW   0x10  /* PC IOP shadow register (PC[23-0])
*/
#define MODE2_SHDW 0x11   /* Mode2 IOP shadow register
(MODE2[31-25])     */
#define EPB2  0x14      /* EXternal port DMA buffer 2
*/
#define EPB3  0x16      /* External port DMA buffer 3
*/
#define BMAX   0x18       /* Bus time-out maxi-
mum                         */
#define BCNT   0x19     /* Bus time-out counter
*/
#define ELAST   0x1b    /* Address of last external access for
page detect */
#define DMAC10 0x1c   /* EP DMA10 control register
*/
#define DMAC11 0x1d   /* EP DMA11 control register
*/
```

```
#define DMAC12 0x1e   /* EP DMA12 control register
*/
#define DMAC13 0x1f   /* EP DMA13 Control register
*/

#define II4   0x30   /* Internal DMA4 memory address
*/
#define IM4    0x31    /* Internal DMA4 memory access modi-
fier         */
#define C4     0x32    /* Contains number of DMA4 transfers
remaining */
#define CP4   0x33   /* Points to next DMA4 parameters
*/
#define GP4   0x34   /* DMA4 General purpose / 2-D DMA
*/
#define DB4   0x35   /* DMA4 General purpose / 2-D DMA
*/
#define DA4   0x36   /* DMA4 General purpose / 2-D DMA
*/

#define DMASTAT 0x37   /* DMA channel status regis-
ter                  */

#define II5   0x38  /* Internal DMA5 memory address
*/
#define IM5    0x39   /* Internal DMA5 memory access modi-
fier          */
#define C5     0x3a   /* Contains number of DMA5 transfers remain-
ing   */
#define CP5   0x3b  /* Points to next DMA5 parameters
*/
#define GP5   0x3c  /* DMA5 General purpose / 2-D DMA
*/
```

```
#define DB5    0x3d   /* DMA5 General purpose / 2-D DMA
*/
#define DA5    0x3e   /* DMA5 General purpose / 2-D DMA
*/

#define II10   0x40   /* Internal DMA10 memory address
*/
#define IM10   0x41   /* Internal DMA10 memory access modifier
*/
#define C10     0x42   /* Contains number of DMA10 transfers
remaining */
#define CP10   0x43   /* Points to next DMA10 parameters
*/
#define GP10   0x44   /* DMA10 General purpose
*/
#define EI10   0x45   /* External DMA10 address
*/
#define EM10   0x46   /* External DMA10 address modifier
*/
#define EC10   0x47   /* External DMA10 counter
*/

#define II11   0x48   /* Internal DMA11 memory address
*/
#define IM11   0x49   /* Internal DMA11 memory access modifier
   */
#define C11     0x4a   /* Contains number of DMA11 transfers
remaining */
#define CP11   0x4b   /* Points to next DMA11 parameters
*/
#define GP11   0x4c   /* DMA11 General purpose
*/
#define EI11   0x4d   /* External DMA11 address
*/
```

```
#define EM11   0x4e   /* External DMA11 address modifier
*/
#define EC11   0x4f   /* External DMA counter
*/

#define II12   0x50   /* Internal DMA12 memory address
*/
#define IM12   0x51   /* Internal DMA12 memory access modifier
*/
#define C12    0x52   /* Contains number of DMA12 transfers
remaining */
#define CP12   0x53   /* Points to next DMA12 parameters
*/
#define GP12   0x54   /* DMA12 General purpose
*/
#define EI12   0x55   /* External DMA12 address
*/
#define EM12   0x56   /* External DMA12 address modifier
*/
#define EC12   0x57   /* External DMA12 counter
*/

#define II13   0x58   /* Internal DMA13 memory address
*/
#define IM13   0x59   /* Internal DMA13 memory access modifier
*/
#define C13    0x5a   /* Contains number of DMA13 transfers
remaining */
#define CP13   0x5b   /* Points to next DMA13 parameters
*/
#define GP13   0x5c   /* DMA13 General purpose
*/
#define EI13   0x5d   /* External DMA13 address
*/
```

```
#define EM13   0x5e   /* External DMA13 address modifier
*/
#define EC13   0x5f   /* External DMA13 counter
*/

#define II0    0x60   /* Internal DMA0 memory address
*/
#define IM0    0x61   /* Internal DMA0 memory access modifier
*/
#define C0     0x62   /* Contains number of DMA0 transfers
remaining   */
#define CP0    0x63   /* Points to next DMA0 parameters      */
#define GP0    0x64   /* DMA0 General purpose / 2-D DMA      */
#define DB0    0x65   /* DMA0 General purpose / 2-D DMA      */
#define DA0    0x66   /* DMA0 General purpose / 2-D DMA      */

#define II1    0x68   /* Internal DMA1 memory address
*/
#define IM1    0x69   /* Internal DMA1 memory access modifier
*/
#define C1     0x6a   /* Contains number of DMA1 transfers
remaining   */
#define CP1    0x6b   /* Points to next DMA1 parameters      */
#define GP1    0x6c   /* DMA1 General purpose / 2-D DMA      */
#define DB1    0x6d   /* DMA1 General purpose / 2-D DMA      */
#define DA1    0x6e   /* DMA1 General purpose / 2-D DMA      */

#define II2    0x70   /* Internal DMA2 memory address
*/
#define IM2    0x71   /* Internal DMA2 memory access modifier
*/
#define C2     0x72   /* Contains number of DMA2 transfers
remaining   */
#define CP2    0x73   /* Points to next DMA2 parameters      */
```

```
#define GP2    0x74   /* DMA2 General purpose / 2-D DMA      */
#define DB2    0x75   /* DMA2 General purpose / 2-D DMA      */
#define DA2    0x76   /* DMA2 General purpose / 2-D DMA      */

#define II3   0x78  /* Internal DMA3 memory address
*/
#define IM3   0x79  /* Internal DMA3 memory access modifier
*/
#define C3     0x7a   /* Contains number of DMA3 transfers
remaining   */
#define CP3    0x7b   /* Points to next DMA3 parameters      */
#define GP3    0x7c   /* DMA3 General purpose / 2-D DMA      */
#define DB3    0x7d   /* DMA3 General purpose / 2-D DMA      */
#define DA3    0x7e   /* DMA3 General purpose / 2-D DMA      */

#define II6   0x80  /* Internal DMA6 memory address
*/
#define IM6   0x81  /* Internal DMA6 memory access modifier
*/
#define C6     0x82 /* Contains number of DMA6 transfers
remaining    */
#define CP6    0x83   /* Points to next DMA6 parameters      */
#define GP6    0x84   /* DMA6 General purpose / 2-D DMA      */
#define DB6    0x85   /* DMA6 General purpose / 2-D DMA      */
#define DA6    0x86   /* DMA6 General purpose / 2-D DMA      */

#define II7   0x88  /* Internal DMA7 memory address
*/
#define IM7   0x89  /* Internal DMA7 memory access modifier
*/
#define C7     0x8a   /* Contains number of DMA7 transfers
remaining   */
#define CP7    0x8b   /* Points to next DMA7 parameters      */
#define GP7    0x8c   /* DMA7 General purpose / 2-D DMA      */
```

```
#define DB7    0x8d   /* DMA7 General purpose / 2-D DMA      */
#define DA7    0x8e   /* DMA7 General purpose / 2-D DMA      */

#define II8   0x90   /* Internal DMA8 memory address
*/
#define IM8   0x91   /* Internal DMA8 memory access modifier
*/
#define C8     0x92   /* Contains number of DMA8 transfers
remaining   */
#define CP8    0x93   /* Points to next DMA8 parameters      */
#define GP8    0x94   /* DMA8 General purpose / 2-D DMA      */
#define DB8    0x95   /* DMA8 General purpose / 2-D DMA      */
#define DA8    0x96   /* DMA8 General purpose / 2-D DMA      */

#define II9   0x98   /* Internal DMA9 memory address
*/
#define IM9   0x99   /* Internal DMA9 memory access modifier
*/
#define C9     0x9a   /* Contains number of DMA9 transfers
remaining   */
#define CP9    0x9b   /* Points to next DMA9 parameters      */
#define GP9    0x9c   /* DMA9 General purpose / 2-D DMA      */
#define DB9    0x9d   /* DMA9 General purpose / 2-D DMA      */
#define DA9    0x9e   /* DMA9 General purpose / 2-D DMA      */

/* Emulation/Breakpoint Registers (remapped from UREG space)
*/
/* NOTES:
       - These registers are ONLY accessible by the core
       - It is *highly* recommended that these facilities be
accessed only
         through the ADI emulator routines
*/
/* Core Emulation HWBD Registers */
```

```
#define PSA1S 0xa0        /* Instruction address start #1 */
#define PSA1E 0xa1        /* Instruction address end   #1 */
#define PSA2S 0xa2        /* Instruction address start #2 */
#define PSA2E 0xa3      /* Instruction address end  #2
*/
#define PSA3S 0xa4      /* Instruction address start #3
*/
#define PSA3E 0xa5      /* Instruction address end  #3
*/
#define PSA4S 0xa6      /* Instruction address start #4
*/
#define PSA4E 0xa7      /* Instruction address end  #4
*/
#define PMDAS 0xa8      /* Program Data address start
*/
#define PMDAE 0xa9      /* Program Data address end
*/
#define DMA1S 0xaa      /* Data address start #1
*/
#define DMA1E 0xab      /* Data address end  #1
*/
#define DMA2S 0xac      /* Data address start #2
*/
#define DMA2E 0xad      /* Data address end  #2
*/
#define EMUN   0xae        /* hwbp hit-count regis-
ter                   */

/* IOP Emulation HWBP Bounds Registers */
#define IOAS   0xb0   /* IOA Upper Bounds Register    */
#define IOAE   0xb1   /* IOA Lower Bounds Register    */
#define EPAS   0xb2   /* EPA Upper Bounds Register    */
#define EPAE   0xb3   /* EPA Lower Bounds Register    */
```

```
#define LBUF0 0xc0   /* Link buffer 0      */
#define LBUF1 0xc2   /* Link buffer 1      */
#define LBUF2 0xc4   /* Link buffer 2      */
#define LBUF3 0xc6   /* Link buffer 3      */
#define LBUF4 0xc8   /* Link buffer 4      */
#define LBUF5 0xca   /* Link buffer 5      */
#define LCTL0 0xcc   /* Link buffer control      */
#define LCTL1 0xcd   /* Link buffer control      */
#define LCOM   0xce   /* Link common control      */
#define LAR    0xcf   /* Link assignment register      */
#define LSRQ   0xd0   /* Link service request and mask register
*/
#define LPATH1 0xd1   /* Link path register 1      */
#define LPATH2 0xd2   /* Link path register 2      */
#define LPATH3 0xd3   /* Link path register 3      */
#define LPCNT 0xd4   /* Link path counter
*/
#define CNST1 0xd5   /* Link port constant 1 register
*/
#define CNST2 0xd6   /* Link port constant 2 register
*/

#define STCTL0 0xe0  /* Serial Port 0 Transmit Control Register
*/
#define SRCTL0 0xe1   /* Serial Port 0 Receive Control Regis-
ter          */
#define TX0    0xe2  /* Serial Port 0 Transmit Data Buffer
*/
#define RX0    0xe3  /* Serial Port 0 Receive Data Buffer
*/
#define TDIV0   0xe4   /* Serial Port 0 Transmit Divi-
sor                    */
#define TCNT0  0xe5  /* Serial Port 0 Transmit Count Reg
*/
```

```
#define RDIV0   0xe6   /* Serial Port 0 Receive Divi-
sor                     */
#define RCNT0  0xe7  /* Serial Port 0 Receive Count Reg
*/
#define MTCS0   0xe8 /* Serial Port 0 Mulitchannel Transmit
Selector      */
#define MRCS0   0xe9 /* Serial Port 0 Mulitchannel Receive Selec-
tor       */
#define MTCCS0 0xea /* Serial Port 0 Mulitchannel Transmit Selec-
tor      */
#define MRCCS0 0xeb /* Serial Port 0 Mulitchannel Receive Selec-
tor       */
#define KEYWD0 0xec /* Serial Port 0 Receive Comparison Register
*/
#define KEYMASK0 0xed   /* Serial Port 0 Receive Comparison Mask
Register */
#define SPATH0 0xee /* Serial Port 0 Path Length (Mesh Multipro-
cessing) */
#define SPCNT0 0xef /* Serial Port 0 Path Counter (Mesh Multipro-
cessing) */

#define STCTL1 0xf0  /* Serial Port 1 Transmit Control Register
*/
#define SRCTL1 0xf1   /* Serial Port 1 Receive Control Regis-
ter           */
#define TX1   0xf2   /* Serial Port 1 Transmit Data Buffer
*/
#define RX1   0xf3   /* Serial Port 1 Receive Data Buffer
*/
#define TDIV1 0xf4    /* Serial Port 1 Transmit Divi-
sor                    */
#define TCNT1 0xf5   /* Serial Port 1 Transmit Count Reg
*/
```

```
#define RDIV1 0xf6    /* Serial Port 1 Receive Divi-
sor                     */
#define RCNT1 0xf7   /* Serial Port 1 Receive Count Reg
*/
#define MTCS1 0xf8    /* Serial Port 1 Mulitchannel Transmit
Selector      */
#define MRCS1 0xf9    /* Serial Port 1 Mulitchannel Receive
Selector       */
#define MTCCS1 0xfa   /* Serial Port 1 Mulitchannel Transmit
Selector      */
#define MRCCS1 0xfb   /* Serial Port 1 Mulitchannel Receive
Selector       */
#define KEYWD1 0xfc /* Serial Port 1 Receive Comparison Register
*/
#define KEYMASK1 0xfd /* Serial Port 1 Receive Comparison Mask
Register    */
#define SPATH1 0xfe   /* Serial Port 1 Path Length (Mesh Multi-
processing) */
#define SPCNT1 0xff   /* Serial Port 1 Path Counter (Mesh Multi-
processing) */

/*-------------------------------------------------------------
---------*/
/*        IOP Register Bit Defini-
tions                                          */
/*-------------------------------------------------------------
---------*/
/* SYSCON Register */
#define SRST 0x00000001 /* Soft Reset
*/
#define BSO   0x00000002 /* Boot Select Override
*/
#define IIVT 0x00000004 /* Internal Interrupt Vector Table
*/
```

```
#define IWT    0x00000008   /* Instruction word transfer (0 =
data, 1 = inst) */
#define HPM000 0x00000000 /* Host packing mode: None
*/
#define HPM001 0x00000010 /* Host packing mode: 16/48
*/
#define HPM010 0x00000020 /* Host packing mode: 16/64
*/
#define HPM011 0x00000030 /* Host packing mode: 32/48
*/
#define HPM100 0x00000040 /* Host packing mode: 32/64
*/
#define HMSWF   0x00000080 /* Host packing order (0 = LSW first,
1 = MSW) */
#define HPFLSH 0x00000100 /* Host pack flush
*/
#define IMDW0   0x00000200    /* Internal memory block 0,
extended data (40 bit) */
#define IMDW1   0x00000400   /* Internal memory block 1, extended
data (40 bit) */
#define ADREDY 0x00000800 /* Active Drive Ready
*/

#define BHD   0x00010000 /* Buffer Hand Disable
*/
#define EBPR00 0x00000000 /* External bus priority: Even
*/
#define EBPR01 0x00020000 /* External bus priority: Core has pri-
ority */
#define EBPR10 0x00040000 /* External bus priority: IO has
priority   */

#define DCPR 0x00080000 /* Select rotating access priority on
DMA10 - DMA13 */
```

```
#define LDCPR 0x00100000    /* Select rotating access priority on
DMA4 - DMA9 */
#define PRROT 0x00200000 /* Select rotating priority between
LPort and EPort */
#define COD   0x00400000 /* Clock Out Dis-
able                                */

#define IMGR 0x10000000 /* Internal memory block grouping (for
the MSP)   */

/* SYSTAT Register */
#define HSTM   0x00000001 /* Host is the Bus Master
*/
#define BSYN    0x00000002 /* Bus arbitration logic is
synchronized     */
#define CRBM   0x00000070 /* Current ADSP21160 Bus Master
*/
#define IDC    0x00000700 /* ADSP21160 ID Code
*/
#define DWPD    0x00001000 /* Direct write pending (0 = none, 1 =
pending) */
#define VIPD   0x00002000 /* Vector interrupt pending (1 = pend-
ing)    */
#define HPS    0x0000c000 /* Host pack status
*/
#define CRAT  0x00070000 /* CLK_CFG(3-0), Core:CLKIN clock ratio
*/

/* WAIT Register */
#define EBOS1   0x00000001 /* External Bank 0 Sync, min 2-cycle
reads,
                           1-cycle writes */
```

```
#define EB0S2   0x00000002 /* External Bank 0 Sync, min 2-cycle
reads,
                           2-cycle writes */
#define EB1S1   0x00000020 /* External Bank 1 Sync, min 2-cycle
reads,
                           1-cycle writes */
#define EB1S2   0x00000040 /* External Bank 1 Sync, min 2-cycle
reads,
                           2-cycle writes */
#define EB2S1   0x00000400 /* External Bank 2 Sync, min 2-cycle
reads,
                           1-cycle writes */
#define EB2S2   0x00000800 /* External Bank 2 Sync, min 2-cycle
reads,
                           2-cycle writes */
#define EB3S1   0x00008000 /* External Bank 3 Sync, min 2-cycle
reads,
                           1-cycle writes */
#define EB3S2   0x00010000 /* External Bank 3 Sync, min 2-cycle
reads,
                           2-cycle writes */
#define UBS1   0x00100000 /* Unbanked Sync, min 2-cycle reads,
1-cycle writes */
#define UBS2   0x00200000 /* Unbanked Sync, min 2-cycle reads,
2-cycle writes */
#define HIDMA 0x80000000 /* Single idle cycle for DMA hand-
shake          */

/* LCTL0 Register */
#define L0EN0x00000001/* LBUF0 Enable*/
#define L0DEN0x00000002/* LBUF0 DMA Enable*/
#define L0CHEN0x00000004/* LBUF0 DMA Chaining Enable*/
#define L0TRAN0x00000008/* LBUF0 Transmit (1=Transmit, 0=Receive)
*/
```

```
#define L0EXT0x00000010/* LBUF0 Extended Word Size
                          (1 = 48-bit, 0 = 32-bit)
*/
#define L0CLKD00x00000020/* LBUF0 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L0CLKD10x00000040/* LBUF0 Clock Divisor 1 (00 = 4)*/
#define L0DMA2D0x00000080/* LBUF0 2-Dimensional DMA Enable*/
#define L0PDRDE0x00000100/* LPORT0 pull-down Resistor Disable*/
#define L0DPWID0x00000200/* LBUF0 Data Path Width (1 = 8-bit, 0 =
4-bit)*/

#define L1EN0x00000400/* LBUF1 Enable*/
#define L1DEN0x00000800/* LBUF1 DMA Enable*/
#define L1CHEN0x00001000/* LBUF1 DMA Chaining Enable*/
#define L1TRAN0x00002000/* LBUF1 Transmit (1=Transmit, 0=Receive)
*/
#define L1EXT0x00004000/* LBUF1 Extended Word Size
                          (1 = 48-bit, 0 = 32-bit)
*/
#define L1CLKD00x00008000/* LBUF1 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L1CLKD10x00010000/* LBUF1 Clock Divisor 1 (00 = 4)*/
#define L1DMA2D0x00020000/* LBUF1 2-Dimensional DMA Enable*/
#define L1PDRDE0x00040000/* LPORT1 pull-down Resistor Disable*/
#define L1DPWID0x00080000/* LBUF1 Data Path Width (1 = 8-bit, 0 =
4-bit)*/

#define L2EN0x00100000/* LBUF2 Enable*/
#define L2DEN0x00200000/* LBUF2 DMA Enable*/
#define L2CHEN0x00400000/* LBUF2 DMA Chaining Enable*/
#define L2TRAN0x00800000/* LBUF2 Transmit (1=Transmit, 0=Receive)
*/
```

```
#define L2EXT0x01000000/* LBUF2 Extended Word Size
                        (1 = 48-bit, 0 = 32-bit)
*/
#define L2CLKD00x02000000/* LBUF2 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L2CLKD10x04000000/* LBUF2 Clock Divisor 1 (00 = 4)*/
#define L2DMA2D0x08000000/* LBUF2 2-Dimensional DMA Enable*/
#define L2PDRDE0x10000000/* LPORT2 pull-down Resistor Disable*/
#define L2DPWID0x20000000/* LBUF2 Data Path Width (1 = 8-bit, 0 =
4-bit)*/

/* LCTL1 Register */
#define L3EN0x00000001/* LBUF3 Enable*/
#define L3DEN0x00000002/* LBUF3 DMA Enable*/
#define L3CHEN0x00000004/* LBUF3 DMA Chaining Enable*/
#define L3TRAN0x00000008/* LBUF3 Transmit (1=Transmit, 0=Receive)
*/
#define L3EXT0x00000010/* LBUF3 Extended Word Size
                        (1 = 48-bit, 0 = 32-bit)
*/
#define L3CLKD00x00000020/* LBUF3 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L3CLKD10x00000040/* LBUF3 Clock Divisor 1 (00 = 4) */
#define L3DMA2D0x00000080/* LBUF3 2-Dimensional DMA Enable  */
#define L3PDRDE0x00000100/* LPORT3 pull-down Resistor Disable */
#define L3DPWID0x00000200/* LBUF3 Data Path Width (1 = 8-bit, 0 =
4-bit)  */

#define L4EN0x00000400/* LBUF4 Enable */
#define L4DEN0x00000800/* LBUF4 DMA Enable */
#define L4CHEN0x00001000/* LBUF4 DMA Chaining Enable */
#define L4TRAN0x00002000/* LBUF4 Transmit (1=Transmit, 0=Receive)
*/
```

```
#define L4EXT0x00004000/* LBUF4 Extended Word Size
                        (1 = 48-bit, 0 = 32-bit)
*/
#define L4CLKD00x00008000/* LBUF4 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L4CLKD10x00010000/* LBUF4 Clock Divisor 1 (00 = 4) */
#define L4DMA2D0x00020000/* LBUF4 2-Dimensional DMA Enable */
#define L4PDRDE0x00040000/* LPORT4 pull-down Resistor Disable */
#define L4DPWID0x00080000/* LBUF4 Data Path Width (1 = 8-bit, 0 =
4-bit)*/

#define L5EN0x00100000/* LBUF5 Enable */
#define L5DEN0x00200000/* LBUF5 DMA Enable */
#define L5CHEN0x00400000/* LBUF5 DMA Chaining Enable */
#define L5TRAN0x00800000/* LBUF5 Transmit (1=Transmit, 0=Receive)
*/
#define L5EXT0x01000000/* LBUF5 Extended Word Size
                            (1 = 48-bit, 0 = 32-bit)*/
#define L5CLKD00x02000000/* LBUF5 Clock Divisor 0 (01 = 1, 10 =
2, 11 = 3) */
#define L5CLKD10x04000000/* LBUF5 Clock Divisor 1 (00 = 4) */
#define L5DMA2D0x08000000/* LBUF5 2-Dimensional DMA Enable */
#define L5PDRDE0x10000000/* LPORT5 pull-down Resistor Disable */
#define L5DPWID0x20000000/* LBUF5 Data Path Width (1 = 8-bit, 0 =
4-bit)*/

/* LCOM Register */
#define L0STAT00x00000001/* LBUF0 Status 0 (11=full, 00=empty) */
#define L0STAT10x00000002/* LBUF0 Status 1 (10=partially full,
                          01=reserved)
*/
#define L1STAT00x00000004/* LBUF1 Status 0 (11=full, 00=empty) */
```

```
#define L1STAT10x00000008/* LBUF1 Status 1 (10=partially full,
                            01=reserved)
*/
#define L2STAT00x00000010/* LBUF2 Status 0 (11=full, 00=empty) */
#define L2STAT10x00000020/* LBUF2 Status 1 (10=partially full,
                            01=reserved)
*/
#define L3STAT00x00000040/* LBUF3 Status 0 (11=full,00=empty) */
#define L3STAT10x00000080/* LBUF3 Status 1 (10=partially full,
                            01=reserved)
*/
#define L4STAT00x00000100/* LBUF2 Status 0 (11=full,00=empty) */
#define L4STAT10x00000200/* LBUF2 Status 1 (10=partially full,
                            01=reserved)
*/
#define L5STAT00x00000400/* LBUF5 Status 0 (11=full,00=empty)
*/
#define L5STAT10x00000800/* LBUF5 Status 1 (10=partially full,
                           01=reserved)
*/
#define LRERR00x04000000/* LBUF0 RxError Status(1=incom-
plete,0=complete) */
#define LRERR10x08000000/* LBUF1 RxError Status(1=incom-
plete,0=complete) */
#define LRERR20x10000000/* LBUF2 RxError Status(1=incom-
plete,0=complete) */
#define LRERR30x20000000/* LBUF3 RxError Status(1=incom-
plete,0=complete) */
#define LRERR40x40000000/* LBUF4 RxError Status(1=incom-
plete,0=complete) */
#define LRERR50x80000000/* LBUF5 RxError Status(1=incom-
plete,0=complete) */

/* LSRQ Register */
```

```
#define L0TM0x00000010/* LPORT0 Transmit Mask*/
#define L0RM0x00000020/* LPORT0 Receive Mask*/
#define L1TM0x00000040/* LPORT1 Transmit Mask*/
#define L1RM0x00000080/* LPORT1 Receive Mask*/
#define L2TM0x00000100/* LPORT2 Transmit Mask*/
#define L2RM0x00000200/* LPORT2 Receive Mask*/
#define L3TM0x00000400/* LPORT3 Transmit Mask*/
#define L3RM0x00000800/* LPORT3 Receive Mask*/
#define L4TM0x00001000/* LPORT4 Transmit Mask*/
#define L4RM0x00002000/* LPORT4 Receive Mask*/
#define L5TM0x00004000/* LPORT5 Transmit Mask*/
#define L5RM0x00008000/* LPORT5 Receive Mask*/
#define L0TRQ0x00100000/* LPORT0 Transmit Request Status*/
#define L0RRQ0x00200000/* LPORT0 Receive Request Status*/
#define L1TRQ0x00400000/* LPORT1 Transmit Request Status*/
#define L1RRQ0x00800000/* LPORT1 Receive Request Status*/
#define L2TRQ0x01000000/* LPORT2 Transmit Request Status*/
#define L2RRQ0x02000000/* LPORT2 Receive Request Status*/
#define L3TRQ0x04000000/* LPORT3 Transmit Request Status*/
#define L3RRQ0x08000000/* LPORT3 Receive Request Status*/
#define L4TRQ0x10000000/* LPORT4 Transmit Request Status*/
#define L4RRQ0x20000000/* LPORT4 Receive Request Status*/
#define L5TRQ0x40000000/* LPORT5 Transmit Request Status*/
#define L5RRQ0x80000000/* LPORT5 Receive Request Status*/

#endif
```

B INTERRUPT VECTOR ADDRESSES

Table B-1 shows all ADSP-21160 DSP interrupts, listed according their bit position in the IRPTL and IMASK registers. For more information, see "Interrupt Latch Register (IRPTL)" on page A-18 and "Interrupt Mask Register (IMASK)" on page A-22. Also shown is the address of the interrupt vector. Each vector is separated by four memory locations. The addresses in the vector table represent offsets from a base address. For an interrupt vector table in internal memory, the base address is 0x0004 0000. For an interrupt vector table in external memory, the base address is 0x0080 0000.

The interrupt name column in Table B-1 lists a mnemonic name for each interrupt as they are defined by the def21160.h file that comes with the software development tools. For more information, see "Register and Bit #Defines File (def21160.h)" on page A-82.

Table B-1. ADSP-21160 Interrupt Vector Addresses

Register	IRPTL/ IMASK, LIRPTL Bit#	Vector Address	Interrupt Name	Function
IRPTL	0	0x00	EMUI	Emulator (read-only, non-maskable) HIGHEST PRIORITY
IRPTL	1	0x04	RSTI	Reset (read-only, non-maskable)
IRPTL	2	0x08	IICDI	Illegal Input Condition Detected
IRPTL	3	0x0C	SOVFI	Status, loop, or mode stack overflow; or PC stack full
IRPTL	4	0x10	TMZHI	Timer=0 (high priority option)

Table B-1. ADSP-21160 Interrupt Vector Addresses (Cont'd)

Register	IRPTL/ IMASK, LIRPTL Bit#	Vector Address	Interrupt Name	Function
IRPTL	5	0x14	VIRPTI	Vector Interrupt
IRPTL	6	0x18	IRQ2I	$\overline{\text{IRQ2}}$ asserted
IRPTL	7	0x1C	IRQ1I	$\overline{\text{IRQ1}}$ asserted
IRPTL	8	0x20	IRQ0I	$\overline{\text{IRQ0}}$ asserted
IRPTL	9	0x24	-	Reserved
IRPTL	10	0x28	SPR0I	DMA Channel 0 - SPORT0 Receive
IRPTL	11	0x2C	SPR1I	DMA Channel 1 - SPORT1 Receive
IRPTL	12	0x30	SPT0I	DMA Channel 2 - SPORT0 Transmit
IRPTL	13	0x34	SPT1I	DMA Channel 3 - SPORT1 Transmit
LIRPTL	0	0x38	LP0I	DMA Channel 4 - Link Buffer 0
LIRPTL	1	0x3C	LP1I	DMA Channel 5 - Link Buffer 1
LIRPTL	2	0x40	LP2I	DMA Channel 6 - Link Buffer 2
LIRPTL	3	0x44	LP3I	DMA Channel 7 - Link Buffer 3
LIRPTL	4	0x48	LP4I	DMA Channel 8 - Link Buffer 4
LIRPTL	5	0x4C	LP5I	DMA Channel 9 - Link Buffer 5
IRPTL	15	0x50	EP0I	DMA Channel 10 - Ext. Port Buffer 0
IRPTL	16	0x54	EP1I	DMA Channel 11 - Ext. Port Buffer 1
IRPTL	17	0x58	EP2I	DMA Channel 12 - Ext. Port Buffer 2
IRPTL	18	0x5C	EP3I	DMA Channel 13 - Ext. Port Buffer 3

Table B-1. ADSP-21160 Interrupt Vector Addresses (Cont'd)

Register	IRPTL/ IMASK, LIRPTL Bit#	Vector Address	Interrupt Name	Function
IRPTL	19	0x60	LSRQI	Link Port Service Request
IRPTL	20	0x64	CB7I	Circular Buffer 7 overflow
IRPTL	21	0x68	CB15I	Circular Buffer 15 overflow
IRPTL	22	0x6C	TMZLI	Timer = 0 (low priority option)
IRPTL	23	0x70	FIXI	Fixed-point overflow
IRPTL	24	0x74	FLTOI	Floating-point overflow exception
IRPTL	25	0x78	FLTUI	Floating-point underflow exception
IRPTL	26	0x7C	FLTII	Floating-point invalid exception
IRPTL	27	0x80	SFT0I	User software interrupt 0
IRPTL	28	0x84	SFT1I	User software interrupt 1
IRPTL	29	0x88	SFT2I	User software interrupt 2
IRPTL	30	0x8C	SFT3I	User software interrupt 3
IRPTL	31	0x90	-	Reserved LOWEST PRIORITY

C NUMERIC FORMATS

The DSP supports the 32-bit single-precision floating-point data format defined in the IEEE Standard 754/854. In addition, the DSP supports an extended-precision version of the same format with eight additional bits in the mantissa (40 bits total). The DSP also supports 32-bit fixed-point formats—fractional and integer—which can be signed (twos-complement) or unsigned.

IEEE Single-Precision Floating-Point Data Format

IEEE Standard 754/854 specifies a 32-bit single-precision floating-point format, shown in Figure C-1. A number in this format consists of a sign bit s, a 24-bit significand, and an 8-bit unsigned-magnitude exponent e.

For normalized numbers, the significand consists of a 23-bit fraction f and a "hidden" bit of 1 that is implicitly presumed to precede f_{22} in the significand. The binary point is presumed to lie between this hidden bit and f_{22}. The least significant bit (LSB) of the fraction is f_0; the LSB of the exponent is e_0.

The hidden bit effectively increases the precision of the floating-point significand to 24 bits from the 23 bits actually stored in the data format. It also insures that the significand of any number in the IEEE normalized number format is always greater than or equal to 1 and less than 2.

The unsigned exponent e can range between $1 \leq e \leq 254$ for normal numbers in the single-precision format. This exponent is biased by +127 (254, 2). To calculate the true unbiased exponent, 127 must be subtracted from e.

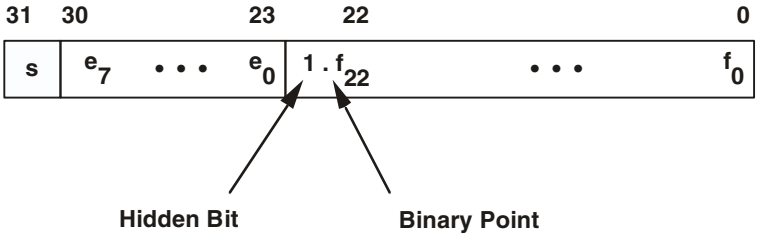


Figure C-1. IEEE 32-Bit Single-Precision Floating-Point Format

The IEEE Standard also provides for several special data types in the single-precision floating-point format:

- An exponent value of 255 (all ones) with a nonzero fraction is a Not-A-Number (NAN). NANs are usually used as flags for data flow control, for the values of uninitialized variables, and for the results of invalid operations such as $0 * \infty$.
- Infinity is represented as an exponent of 255 and a zero fraction. Note that because the fraction is signed, both positive and negative Infinity can be represented.
- Zero is represented by a zero exponent and a zero fraction. As with Infinity, both positive Zero and negative Zero can be represented.

The IEEE single-precision floating-point data types supported by the DSP and their interpretations are summarized in Table C-1.

Table C-1. IEEE Single-Precision Floating-Point Data Types

Type	Exponent	Fraction	Value
NAN	255	Nonzero	Undefined
Infinity	255	0	$(-1)^s$ Infinity
Normal	$1 \leq e \leq 254$	Any	$(-1)^s (1.f_{22-0}) 2^{e-127}$
Zero	0	0 $(-1)^s$ Zero	

Extended-Precision Floating-Point Format

The extended-precision floating-point format is 40 bits wide, with the same 8-bit exponent as in the standard format but a 32-bit significand. This format is shown in Figure C-2. In all other respects, the extended-precision floating-point format is the same as the IEEE standard format.

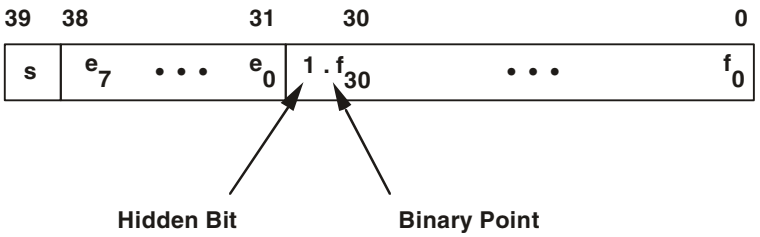


Figure C-2. 40-Bit Extended-Precision Floating-Point Format

Short Word Floating-Point Format

The DSP supports a 16-bit floating-point data type and provides conversion instructions for it. Figure C-3 shows the short float data format has an 11-bit mantissa with a four-bit exponent plus sign bit. The 16-bit floating-point numbers reside in the lower 16 bits of the 32-bit floating-point field.

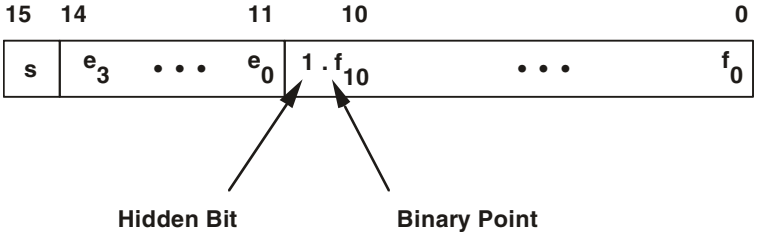


Figure C-3. 16-Bit Floating-Point Format

Packing for Floating-Point Data

Two shifter instructions, FPACK and FUNPACK, perform the packing and unpacking conversions between 32-bit floating-point words and 16-bit floating-point words. The FPACK instruction converts a 32-bit IEEE floating-point number to a 16-bit floating-point number. FUNPACK converts the 16-bit floating-point numbers back to 32-bit IEEE floating-point. Each instruction executes in a single cycle.

The results of the FPACK and FUNPACK operations appear in Table C-2 and Table C-3.

Table C-2. FPACK Operations

Condition	Result
135 < exp	Largest magnitude representation.
120 < exp ≤ 135	Exponent is MSB of source exponent concatenated with the three LSBs of source exponent. The packed fraction is the rounded upper 11 bits of the source fraction.
109 < exp ≤ 120	Exponent=0. Packed fraction is the upper bits (source exponent – 110) of the source fraction prefixed by zeros and the "hidden" 1. The packed fraction is rounded.
exp < 110	Packed word is all zeros.
exp = source exponent **sign bit remains the same in all cases**	

Table C-3. FUNPACK Operations

Condition	Result
0 < exp ≤ 15	Exponent is the 3 LSBs of the source exponent prefixed by the MSB of the source exponent and four copies of the complement of the MSB. The unpacked fraction is the source fraction with 12 zeros appended.
exp = 0	Exponent is (120 – N) where N is the number of leading zeros in the source fraction. The unpacked fraction is the remainder of the source fraction with zeros appended to pad it and the "hidden" 1 stripped away.
exp = source exponent **sign bit remains the same in all cases**	

The short float type supports gradual underflow. This method sacrifices precision for dynamic range. When packing a number which would have underflowed, the exponent is set to zero and the mantissa (including "hidden" 1) is right-shifted the appropriate amount. The packed result is a denormal which can be unpacked into a normal IEEE floating-point number. During the FPACK operation, an overflow will set the SV condition and

non-overflow will clear it. During the FUNPACK operation, the SV condition will be cleared. The SZ and SS conditions are cleared by both instructions.

Fixed-Point Formats

The DSP supports two 32-bit fixed-point formats: fractional and integer. In both formats, numbers can be signed (twos-complement) or unsigned. The four possible combinations are shown in Figure C-4. In the fractional format, there is an implied binary point to the left of the most significant magnitude bit. In integer format, the binary point is understood to be to the right of the LSB. Note that the sign bit is negatively weighted in a twos-complement format.

ALU outputs always have the same width and data format as the inputs. The multiplier, however, produces a 64-bit product from two 32-bit inputs. If both operands are unsigned integers, the result is a 64-bit unsigned integer. If both operands are unsigned fractions, the result is a 64-bit unsigned fraction. These formats are shown in Figure C-5.

If one operand is signed and the other unsigned, the result is signed. If both inputs are signed, the result is signed and automatically shifted left one bit. The LSB becomes zero and bit 62 moves into the sign bit position. Normally bit 63 and bit 62 are identical when both operands are signed. (The only exception is full-scale negative multiplied by itself.) Thus, the left shift normally removes a redundant sign bit, increasing the precision of the most significant product. Also, if the data format is fractional, a single-bit left shift renormalizes the MSP to a fractional format. The signed formats with and without left shifting are shown in Figure C-6.

The multiplier has an 80-bit accumulator to allow the accumulation of 64- bit products. For more information on the multiplier and accumulator, see “Multiply—Accumulator (Multiplier)” on page -13.

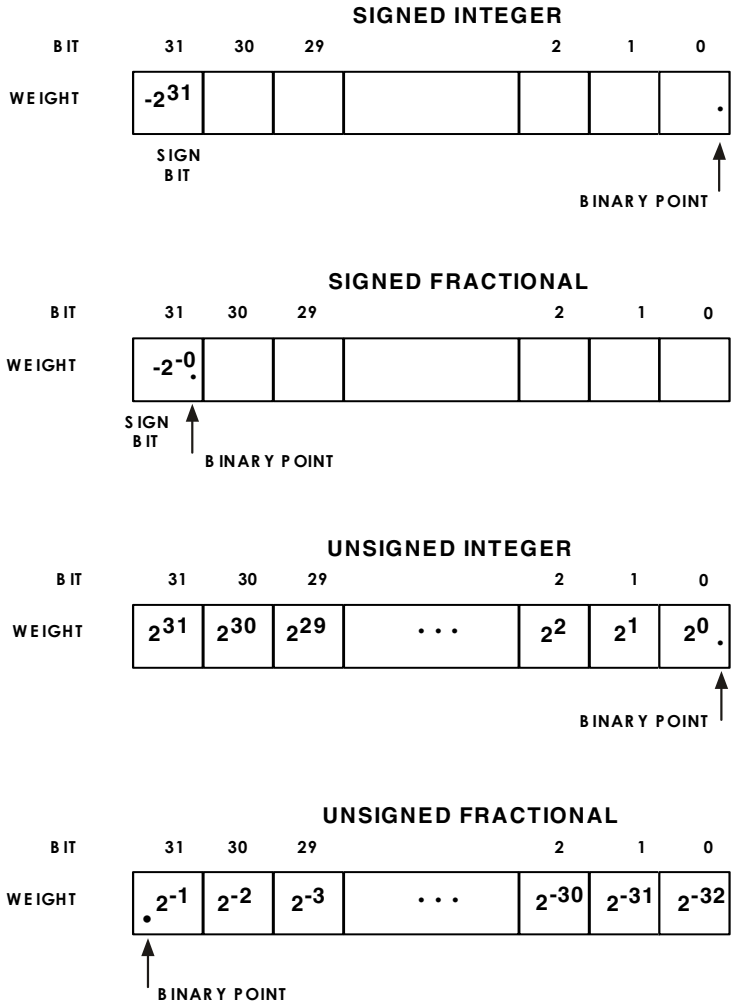


Figure C-4. 32-Bit Fixed-Point Formats

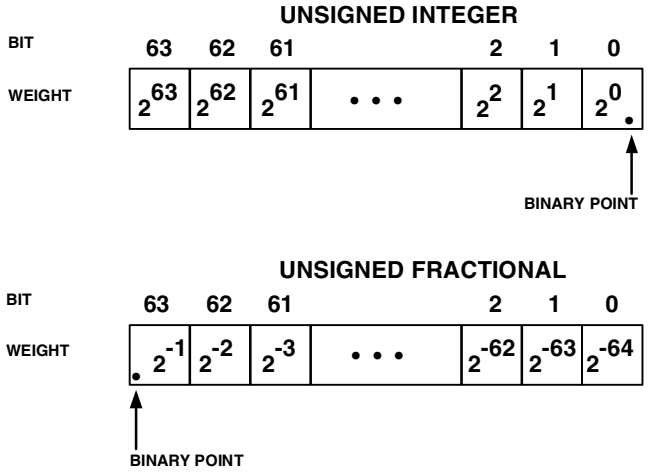


Figure C-5. 64-Bit Unsigned Fixed-Point Product

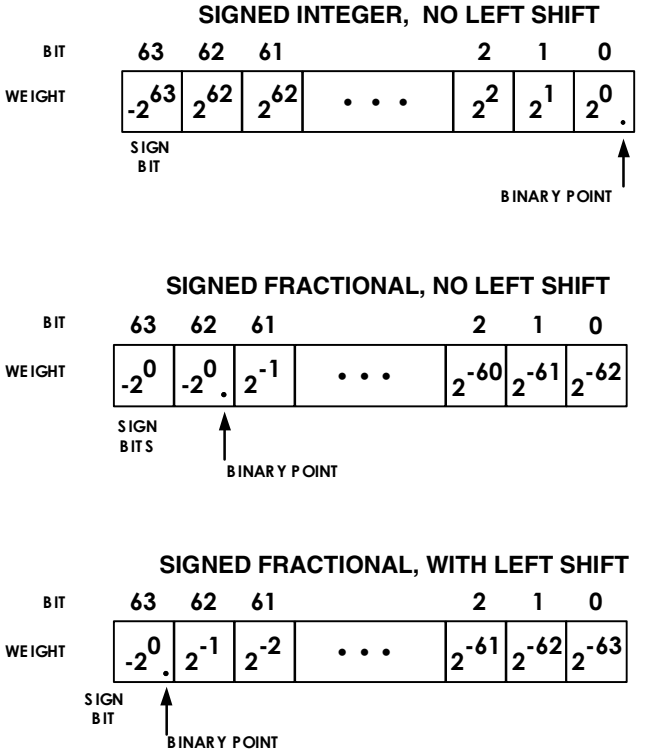


Figure C-6. 64-Bit Signed Fixed-Point Product

G GLOSSARY

These terms are important for understanding DSP architecture.

Arithmetic Logic Unit (ALU)

This part of a processing element performs arithmetic and logic operations on fixed-point and floating-point data.

Asynchronous transfers

These are asynchronous host accesses of the DSP. After acquiring control of the DSP's external bus, the host must assert the $\overline{\text{CS}}$ pin of the DSP it wants to access.

Base address

This is the starting address of a circular buffer to which the DAG wraps around. This address is stored in a DAG Bx register.

Base register

A base (Bx) register is a Data Address Generator (DAG) register that sets up the starting address for a circular buffer.

Bit-reverse addressing

The Data Address Generator (DAG) provides a bit-reversed address during a data move without reversing the stored address.

Boot Memory Space

The DSP supports an external boot EPROM mapped to external memory and selected with the $\overline{\text{BMS}}$ pin. The boot EPROM provides one of the

methods for automatically loading a program into the internal memory of the DSP after power-up or after a software reset.

Broadcast data moves

The Data Address Generator (DAG) performs dual data moves to complementary registers in each processing element to support SIMD mode.

Burst transfers

These multi-cycle synchronous transfers contain a packet of at least two 64-bit transfers. For a DSP master, only a DMA channel can master a burst transaction. As a slave, DSP supports burst read transfers from internal memory, or the EPBx data buffers.

Bus slave or slave mode

A DSP can be a bus slave to another DSP or to a host processor. The DSP becomes a host bus slave when the $\overline{\text{HBG}}$ signal is returned.

Bus transition cycle (BTC)

This cycle passes control of the external bus from one DSP to another (in a multiprocessor system).

Circular buffer addressing

The DAG uses the Ix, Mx, Lx, and Bx register settings to constrain addressing to a range of addresses. This range contains data that the DAG steps through repeatedly, "wrapping around" to repeat stepping through the range of addresses in a circular pattern.

Cluster multiprocessing

This is a multiprocessing system architecture in which the DSP uses its link ports and external port for inter-DSP communication.

Companding (compressing/expanding)

This is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent.

Conditional branches

These are Jump or Call/return instructions whose execution is based on testing an If condition.

Conflict resolution ratio

Because the external port must arbitrate accesses over three internal buses to one external bus, there is a 3:1 conflict resolution ratio at the external port interface. This ratio plus the 2:1 or greater clock ratio between the DSP's internal clock and the external system clock forces systems that fetch instructions or data through the external port must tolerate at least one cycle—and usually many additional cycles—of latency.

Data Address Generators

Data address generators (DAGs) provide memory addresses when data is transferred between memory and registers.

Data flow multiprocessing

This is a multiprocessor system architecture in which the DSP uses its link ports for inter-DSP communication.

Data register file

This is the set of data registers that transfer data between the data buses and the computation units. These registers also provide local storage for operands and results.

Data registers (Dreg)

These are registers in the PEx and PEy processing elements. These registers are hold operands for multiplier, ALU, or shifter operations and are

denoted as Rx when used for fixed point operations or Fx when used for floating-point operations.

Deadlock Resolution

When both the DSP subsystem and the system try to access each other's bus in the same cycle, a deadlock may prevent the completion of either access. Techniques for resolving deadlock vary with the interface: DRAM, host, or multiprocessor DSP.

Delayed branches

These are Jumps and Call/return instructions with the delayed branches (DB) modifier. In delayed branches, no instruction cycles are lost in the pipeline, because the DSP executes the two instructions after the branch while the pipeline fills with instructions from the new branch.

Direct branches

These are Jump or Call/return instructions that use an absolute—not changing at runtime—address (such as a program label) or use a PC-relative address.

Direct reads and writes

These are direct accesses of the DSP's internal memory or I/O processor registers by another DSP or by a host processor.

DMA (Direct Memory Accessing)

The DSP's I/O processor supports DMA of data between DSP memory and external memory, host, or peripherals through the external, link, and serial ports. Each DMA operation transfers an entire block of data.

DMA chaining

The DSP supports chaining together multiple DMA sequences. In chained DMA, the I/O processor loads the next Transfer Control Block

(DMA parameters) into the DMA parameter registers when the current DMA finishes and auto-initializes the next DMA sequence.

DMA parameter registers

These registers function similarly to data address generator registers by setting up a memory access process. These registers include Internal Index registers (IIx), Internal Modify registers (IMx), Count registers (Cx), Chain Pointer registers (CPx), General Purpose registers (GPx), Dimension A and B registers (DAx and DBx), External Index registers (EIx), External Modify registers (EMx), and External Count registers (ECx).

DMA TCB chain loading

This is the process that the I/O processor uses for loading the TCB of the next DMA sequence into the parameter registers during chained DMA.

DMACx control registers

These are the DMA control registers for the EPBx external port buffers: DMAC10, DMAC11, DMAC12, and DMAC13. These corresponding respectively to EPB0, EPB1, EPB2, and EPB3.

Edge-sensitive interrupt

The DSP detects this type of interrupt if the input signal is high (inactive) on one cycle and low (active) on the next cycle when sampled on the rising edge of CLKIN.

Endian Format, Little Versus Big

The DSP uses big-endian format—moves data starting with most-significant-bit and finishing with least-significant-bit—in almost all instances. The two exceptions are bit order for data transfer through the serial port and word order for packing through the external port. For compatibility with little-endian (least-significant-first) peripherals, the DSP supports both big- and little-endian bit order data transfers. Also for compatibility

little endian hosts, the DSP supports both big- and little endian word order data transfers.

Explicit Versus Implicit operations

In SIMD mode, identical instructions execute on the PEx and PEy computational units; the difference is the data. The data registers for PEy operations are identified (implicitly) from the PEx registers in the instruction. This implicit relation between PEx and PEy data registers corresponds to complementary register pairs.

External bus

The DSP extends the following signals off-chip as an external bus: DATA63-0, ADDR31-0, $\overline{\text{RDH}}$, $\overline{\text{RDL}}$, $\overline{\text{WRH}}$, $\overline{\text{WRL}}$, $\overline{\text{MS3-0}}$, $\overline{\text{BMS}}$, CLKOUT, PAGE, BRST, ACK, and $\overline{\text{SBTS}}$.

External memory space

This space ranges from address 0x0080 0000 through 0xFFFF FFFF (Normal word). External memory space refers to the off-chip memory or memory mapped peripherals that are attached to the DSP's external address (ADDR31-0) and data (DATA63-0) buses.

External port FIFO buffers (EPB0, EPB1, EPB2, and EPB3)

These are the I/O processor registers used for external port DMA transfers and single-word data transfers (from other DSPs or from a host processor). These buffers are eight-deep FIFOs.

External port

This port extends the DSPs internal address and data buses off-chip, providing the processor's interface to off-chip memory and peripherals.

Field deposit (Fdep) instructions

These shifter instructions take a group of bits from the input register (starting at the LSB of the 32-bit integer field) and deposit the bits as directed anywhere within the result register.

Field extract (Fext) instructions

These shifter instructions extract a group of bits as directed from anywhere within the input register and place them in the result register (aligned with the LSB of the 32-bit integer field).

Flag update

The DSP's update to status flags occurs at the end of the cycle in which the status is generated and is available on the next cycle.

Harvard architecture

DSPs use memory architectures that have separate buses for program and data storage. The two buses let the DSP get a data word and an instruction simultaneously.

Hold time cycle

This is an inactive bus cycle that the DSP automatically generates at the end of a read or write (depending on the external port access mode) to allow a longer hold time for address and data. The address—and data, if a write—remains unchanged and is driven for one cycle after the read or write strobes are deasserted.

Host transition cycle (HTC)

This cycle passes control of the external bus from the DSP to the host processor. During this cycle the DSP stops driving the $\overline{\text{RDH/L}}$, $\overline{\text{WRH/L}}$, ADDR31-0, $\overline{\text{MS3-0}}$, CLKOUT, PAGE, $\overline{\text{PA}}$, and $\overline{\text{DMAGx}}$ signals, which must then be driven by the host.

I/O processor register

This is one of the control, status, or data buffer registers of the DSP's on-chip I/O processor.

Idle cycle

This is an inactive bus cycle that the DSP automatically generates (depending on the external port access mode) to prevent data bus driver conflicts. Such a conflict can occur when a device with a long output disable time continues to drive after $\overline{\text{RDH/L}}$ is deasserted while another device begins driving on the following cycle.

Idle

This instruction causes the processor to cease operations and hold its current state until an interrupt occurs. Then, the processor services the interrupt and continues normal execution.

Index register

An index register is a Data Address Generator (DAG) register that holds an address and acts as a pointer to memory.

Indirect branches

These Jump or Call/return instructions use a dynamic—changes at runtime—address that comes from the PM data address generator.

Interleaved data

To take advantage of the DSP's data accesses to 4- and 3-column locations, programs must adjust the interleaving of data into (not necessarily sequential) memory locations to accommodate the memory access mode.

Internal memory space

This space ranges from address 0x0000 0000 through 0x0007 FFFF (Normal word). Internal memory space refers to the DSP's on-chip SRAM and memory mapped registers.

Interrupts

These subroutines enable a runtime event (not an instruction) to trigger the execution of the routine.

JTAG port

This port supports the IEEE standard 1149.1 Joint Test Action Group (JTAG) standard for system test. This standard defines a method for serially scanning the I/O status of each component in a system.

Jumps

Jumps transfer program flow permanently to another part of program memory.

Link ports

The DSP has six 8-bit wide link ports, which can connect to other DSPs' or peripherals' link ports. These bidirectional ports have eight data lines, an acknowledge, and a clock line.

Length register

This Data Address Generator (DAG) register sets up the range of addresses a circular buffer.

Level-sensitive interrupt

The DSP detects this type of interrupt if the signal input is low (active) when sampled on the rising edge of `CLKIN`.

Link ports

The DSP has six 10-bit link ports that provide additional I/O capabilities. Link port I/O is especially useful for point-to-point interprocessor communication in multiprocessing systems.

Loop

A loop is one sequence of instructions that executes several times with zero overhead.

Memory Access Modes

The DSP supports Asynchronous and Synchronous modes for accessing external memory space. In asynchronous access mode, the DSP's $\overline{\texttt{RDH/L}}$ and $\overline{\texttt{WRH/L}}$ strobes change before CLKOUT's edge. In synchronous access mode, the DSP's $\overline{\texttt{RDH/L}}$ and $\overline{\texttt{WRH/L}}$ strobes change on CLKOUT's edge.

Memory blocks and banks

The DSP's internal memory is divided into **blocks** that are each associated with different data address generators. The DSP's external memory spaces is divided into **banks**, which may be addressed by either data address generator. External memory banks also may be configured for size and access waitstates.

Modified addressing

The DAG generates an address that is incremented by a value or a register.

Modify address

The Data Address Generator (DAG) increments the stored address without performing a data move.

Modify register

This Data Address Generator (DAG) register provides the increment or step size by which an index register is pre- or post-modified during a register move.

Multifunction computations

Using the many parallel data paths within its computational units, the DSP supports parallel execution of multiple computational instructions. These instructions complete in a single cycle, and they combine parallel operation of the multiplier and the ALU or dual ALU functions. The multiple operations perform the same as if they were in corresponding single-function computations.

Multiplier

This part of a processing element does floating-point and fixed-point multiplication and executes fixed-point multiply/add and multiply/subtract operations.

Multiprocessor memory space

This portion of the DSP's memory map includes the internal memory and I/O processor registers of each DSP in a multiprocessing system. This address space is mapped into the unified address space of the DSP.

Multiprocessor system

This system comprises multiple DSPs, with or without a host processor. The DSPs are connected by the external bus and/or link ports.

Multiprocessor vector interrupt

The vector interrupt (`VIRPT`) permits interprocessor commands to be passed in multiple-processor systems. One DSP writes a vector address to another DSP's `VIRPT` register. Writing the address initiates the vector

interrupt DSP that receives the write, and that DSP executes (vectors to) the interrupt service routine at that address.

Neighbor Registers

In Long word addressed accesses, the DSP moves data to or from two neighboring data registers. The least-significant-32-bits moves to or from the explicit (named) register in the neighbor register pair. In forced Long word accesses (Normal word address with `LW` mnemonic), the DSP converts the Normal word address to Long word, placing the even Normal word location in the explicit register and the odd Normal word location in the other register in the neighbor pair.

Peripherals

Peripherals refer to everything outside the processor core. The ADSP-21160's peripherals include internal memory, external port, I/O processor, JTAG port, and any external devices that connect to the DSP.

Precision

The precision of a floating-point number depends on the number of bits after the binary point in the storage format for the number. The DSP supports two high precision floating-point formats: 32-bit IEEE single-precision floating-point (which uses 8 bits for the exponent and 24 bits for the mantissa) and a 40-bit extended precision version of the IEEE format.

Post-modify addressing

The Data Address Generator (DAG) provides an address during a data move and auto-increments the stored address for the next move.

Pre-modify addressing

The Data Address Generator (DAG) provides a modified address during a data move without incrementing the stored address.

Registers swaps

This special type of register-to-register move instruction uses the special swap operator, <->. A register-to-register swap occurs when registers in different processing elements exchange values.

Saturation (ALU saturation mode)

In this mode, all positive fixed-point overflows return the maximum positive fixed-point number (0x7FFF FFFF), and all negative overflows return the maximum negative number (0x8000 0000).

Semaphore

This flag can be read and written by any of the processors sharing the resource. Semaphores can be used in multiprocessor systems to allow the processors to share resources such as memory or I/O. The value of the semaphore tells the processor when it can access the resource. Semaphores are also useful for synchronizing the tasks being performed by different processors in a multiprocessing system.

Serial ports

The DSP has two synchronous serial ports that provide an inexpensive interface to a wide variety of digital and mixed-signal peripheral devices.

SHARC

This is an acronym for Super Harvard Architecture. This DSP architecture balances a high performance processor core with high performance buses (PM, DM, IO).

Shifter

This part of a processing element completes logical shifts, arithmetic shifts, bit manipulation, field deposit, and field extraction operations on 32-bit operands. Also, the Shifter can derive exponents.

Subroutines

These instructions perform a specific task from another part of program memory. They are executed when the processor temporarily interrupts sequential flow of the main routine. When the task is completed, flow returns to the proper place in the main routine.

Single-word data transfers

These reads and writes to the EPBx external port buffers are performed externally by the DSP bus master (or host) or internally by the DSP slave's core. These occur when DMA is disabled in the DMACx control register.

Synchronous transfers

These are synchronous host accesses of the DSP. $\overline{\text{CS}}$ is not asserted and the host must act like another DSP in a multiprocessor system, by generating an address in multiprocessor memory space, asserting $\overline{\text{PA}}$ and $\overline{\text{WRH/L}}$ or $\overline{\text{RDH/L}}$, and driving out or latching in the data.

TCB chain loading

In this process the DSP's DMA controller downloads a Transfer Control Block from memory and autoinitializes the DMA parameter registers.

Time Division Multiplexed (TDM) mode

The serial ports support TDM or multichannel operations. In multichannel mode, each data word of the serial bit stream occupies a separate channel— each word belongs to the next consecutive channel so that, for example, a 24-word block of data contains one word for each of 24 channels.

Transfer control block (TCB)

This set of DMA parameter register values stored in memory are downloaded by the DSP's DMA controller for chained DMA operations.

Tristate Versus Three-state

Analog Devices documentation uses the term “three-state” instead of “tristate” because Tristate™ is a trademarked term, which is owned by National Semiconductor.

Universal registers (Ureg)

These are any processing element registers (data registers), any Data Address Generator (DAG) registers, any program sequencer registers, and any I/O processor registers.

Von Neumann architecture

This is the architecture used by most (non-DSP) microprocessors. This architecture uses a single address and data bus for memory access.

Waitstates

The DSP applies waitstates to each external memory access, depending on the bank’s external memory access mode (EBxAM). The External Bank Waitstate (EBxWS) field in the WAIT register sets the number of waitstates for each bank.

I INDEX

Symbols

.D unit (*See DAGs or ALU*)
.L unit (*See ALU*)
.M unit (*See Multiplier*)
.S unit (*See Shifter*)

Numerics

16-bit data (*See Short word*)
16-Bit Floating-Point Format C-4
32-bit data (*See Normal word*)
32-Bit Single-Precision Floating-Point Format C-2
40-bit data (*See Extended precision normal word*)
40-Bit Extended-Precision Floating-Point Format C-3
48-bit data (*See Instruction word*)
64-bit data (*See Long word*)

A

Abs function 2-8
Absolute address 3-17, G-4
Access DSP bus 7-84
Access word size 5-40
Accumulator 1-7
Acknowledge (ACK) pin 5-36, 6-30, 6-32, 7-7, 7-49, 7-65, 7-95, 11-5, 11-11
Acknowledge controls 1-13
Acquiring the bus 7-55
Active drive 7-5, 7-53
Active Drive REDY (ADREDY) bit 7-53, 7-61, A-46
Add instructions 2-35
Add with carry 2-8
Add/multiply 2-1, G-11
Add/subtract 2-8
Address bus (ADDR31-0) pins 7-5, 7-46, 7-59, 7-95, 7-105, 11-3, 11-11
address buses 1-3
address fields A-28
Addressing (*See modify, post-modify, pre-modify, bit-reverse, or circular buffer*) (*See also Data Address Generators (DAGs) and broadcast load*)
Adobe Acrobat 1-25
A-law companding 1-15, 9-1, 9-19, 9-20, 9-33
Aliased memory 5-16

Alternate registers (*See Secondary registers*)
ALU Carry (AC) bit 2-10, 3-55, A-9
ALU fixed-point Overflow (AOS) bit 2-10, A-13
ALU Floating-point (AF) bit 2-10, A-11
ALU floating-point Invalid (AI) bit 2-10, A-10
ALU floating-point Invalid Status (AIS) bit 2-10, A-13
ALU floating-point Overflow Status (AVS) bit 2-10, A-13
ALU floating-point Underflow Status (AUS) bit 2-10, A-13
ALU Negative (AN) bit 2-10, A-9
ALU Overflow (AV) bit 2-9, 3-55, A-9
ALU Saturation (ALUSAT) bit 2-2, 2-9, A-4
ALU x-input Sign (AS) bit 2-10, A-9
ALU Zero (AZ) bit 2-9, A-9
Analog Devices product information 1-23
AND breakpoints (ANDBKP) bit 10-10
And, logical 2-8
ANDBKP bit 10-10
Arithmetic Logic Unit (ALU) 1-7, 2-1, 2-7
  Instructions 2-8, 2-10
  Interrupts 3-43
  Operations 2-8
  Saturation 2-9
  Status 2-2, 2-7, 2-9, 2-10, 2-18, 3-43
Arithmetic operations 1-6, 2-7, 2-9
Arithmetic shifts 2-1, G-13
Arithmetic Status (ASTATx and ASTATy) registers 2-7, A-8
Assembler 1-18
Assembly language 2-2
Assign x Link Buffer/port (AxLB) bits 6-43, 6-44, 6-62, A-67
Asymmetric data moves 2-38
Asynchronous access mode 5-36, 7-5, 7-6, 7-13, 7-18, 7-53, 7-60, 7-101, 11-10, G-10
  Direct read 7-72
  Direct write 7-71
  Host interface 7-51
  Interface timing 7-19
  Read/Write—bus slave 7-19
  Read—bus master 7-21
  Slave write FIFO 7-60
  Starting a transfer 7-59
  Write—bus master 7-22
Asynchronous communications 9-5
Asynchronous transfers 7-54, 7-59, G-1
Audience (intended) 1-1
Average instructions 2-8, 2-35

B

Background registers (*See Secondary registers*)

Banks of memory 5-4, 5-21, 5-33, 7-50, G-10
Barrel-shifter (*See Shifter*)
Base (Bx) registers 4-3, 4-17, A-33, G-1
BCNT bus counter register A-54
BHD bit 10-11
BHO bit 10-11
Big-endian format 1-15
Binary log (floating-point operation) 2-8
BIST 10-37
Bit manipulation 2-1, 2-21, G-13
Bit manipulation instruction 3-6
Bit test flag (BTF) bit 3-54, A-12
Bit Tst instruction 2-7
Bit XOR instruction 3-54
Bit-reverse (Bitrev) instruction 4-8, 4-18, 4-26
Bit-reverse addressing 4-1, 4-4, 4-8, A-3, G-1
Bit-reverse addressing (BRx) bits 4-4, 4-8, A-3
Bits
  ANDBKP 10-10
  BHD 10-11
  CADIS 3-13
  CAFRZ 3-13
  COMHALT 10-12
  CSEL 3-57
  DMDSEL 10-10
  EIRQENA 10-8
  EMUENA 10-8
  EMUREADY 10-12
  EMUSPACE 10-12
  ENBRKOUT 10-8
  ENBx 10-10
  EPHALT 10-12
  EPSTOP 10-9
  INIDLE 10-12
  IOSTOP 10-8
  IRPTEN 3-42
  IRQ0E 3-41
  IRQ1E 3-41
  IRQ2E 3-41
  LPISUMI 3-43, A-20
  LPSUM 3-43
  LRU 3-11
  LSEM 3-31
  LSOV 3-31
  LxDPWID bit 8-7
  MTST 10-11
  NEGx 10-9
  NESTM 3-46, A-4
  NOBOOT 10-11
  PC 3-53
  PCEM 3-53
  SS 10-8
  SSEM 3-44
  SSOV 3-44
  STATx 10-13
  STKYy 3-43
  SYSRST 10-8
  TIMEN 3-51
  TMODE 10-11
  VIPD 3-49
  xMODE 10-10
BKSTOP bit 10-8

Blocks of memory 5-1, 5-3, 5-4, 5-16, 5-32, G-10
BMAX bus counter register A-54
Boot memory 5-21, 5-29, G-1
Boot Memory Select (BMS) pin 5-30, 6-16, 6-77, 6-89, 7-11, 11-8
Boot Select Override (BSO) bit 5-28, 5-30, 6-14, 6-16, A-45
Booting 1-14, 5-11, 5-31, 11-48
  Another DSP 7-116
  EPROM 6-78
  External port 6-76
  Link port 6-88, 6-89, 8-5
  Mode selection 6-77, 6-89
  Multiple DSPs 11-48
Boundary
  register 10-17
  scan 10-1, 10-38
Branches and sequencing 3-15
Branching execution 3-15
  Delayed branch 3-16
  Direct and indirect branches 3-16
  Immediate branches 3-18
Breakpoint Output (BRKOUT) pin 10-8
breakpoint registers 10-13
Breakpoint Status (STATx) bit 10-13
Breakpoint Status shift (BRKSTAT) register 10-12
Breakpoint Stop (BKSTOP) bit 10-8
Breakpoint triggering Mode (xMODE) bit 10-10
BRKOUT pin 10-8
Broadcast load 4-1, 4-3, 4-4, 4-5, 5-44, A-5, G-2
  Extended precision normal word data 5-74
  Long word data 5-74
  Normal word data 5-74
  Short word data 5-74
Broadcast load enable (BDCSTx) bits 4-3, 4-4, 4-5, 5-29, 5-34, A-5
Broadcast write 5-19, 7-60, 7-73, 7-118
Broadcast write timing 7-119
BSDL file 10-4
BSDL Reference Guide 10-38
Buffer Hang Disable (BHD) bit 1-21, 6-14, 6-18, 10-11, A-46
Buffer Hang Override (BHO) bit 10-11
Buffer overflow 4-16
  circular 4-9, 4-12
Buffer status 6-58, 6-64
built-in self-test operation (BIST) 10-37
Burst transfer (BRST) pin 7-7, 7-46, 7-95, 7-105, 11-4, 11-11
Burst transfers 6-26, 7-34, 7-36, 7-67, G-2
Bus arbitration 7-103
  Arbitration timing 7-104
  Multiprocessor arbitration 7-101

Bus Arbitration Priority (RPBA) pin 7-107
bus contention A-34
Bus deadlock 7-53, 7-87
  DSP resolution 7-17
  Host resolution 7-89
Bus exchange (*See Program memory bus Exchange (PX) register*)
Bus lock 7-90, 7-99, 7-122
Bus Lock (BUSLK) bit 7-109, 7-122, A-7
Bus master 7-53, 7-84, 7-105
  Acquiring the bus 7-55
  Timeout 7-108
bus master A-4
Bus master (Bm) condition 3-56, 7-103, 7-122
Bus master Count (BCNT) register 7-109
Bus master Max time-out (BMAX) register 7-109, A-53
Bus request timing 7-105
Bus Request, multiprocessor ($\overline{\text{BRx}}$) pins 7-95, 7-101, 7-122, 11-6
Bus slave 7-84, 7-105, G-2
Bus synchronization 7-113, 7-115
Bus Synchronized (BSYN) bit 7-114, 7-125, A-51
Bus Transition Cycle (BTC) 7-25, 7-55, 7-103, G-2
Buses 1-3, 1-11
  Addressing operations 5-4
  Arbitration 5-4
  Data access types 5-40
  Enhancements 1-20
  Priority 5-33
  Processor core 1-20
Bypass capacitors 11-45

C

Cache
  using 3-12
Cache Disable (CADIS) bit 3-13, A-7
Cache efficiency 3-13
Cache Freeze (CAFRZ) bit 3-13, A-7
Cache hit/miss (*See Cache efficiency*)
Call instructions 3-15
  Conditional branch 3-16
  Delayed branch 3-16
Capacitive loading 11-41
Capacitors, bypass 11-45
CCITT G.711 specification 9-20
Chain insertion mode 6-59, 6-74
Chain Pointer (CPx) registers 6-6, 6-9, 6-70, A-59, G-5
Chained DMA
  External port 6-21
  Link ports 6-48
  Serial ports 6-52, 9-18
Chained DMA Enable, external port (CHEN) bit 6-15, A-55
Chained DMA sequences 6-70
Channel current, serial (CHNL) bit 9-12, A-73
Chip Select ($\overline{\text{CS}}$) pin 7-53, 7-54, 7-84, 7-95, 11-6

Circular buffer addressing 1-10, 4-3, 4-4, 4-12, G-2
Registers 4-16
Setup 4-14
SIMD and Long word accesses 4-18
Wrap around 4-16
circular buffer addressing A-5, A-33
Circular buffer addressing Enable (CBUFEN) bit 1-20, 4-3, 4-4, 4-15, A-5
circular buffer x overflow interrupt (CBxI) bit A-21
Circular Buffer x overflow Status (CBxS) bit A-14
Clear interrupt (CI) Jump 3-16
Clear, bit 2-21
Clip function 2-8
CLKOUT disable (COD) bit A-47
Clock derivation 11-14
Clock distribution 11-38
Clock frequencies, serial port 9-3
Clock Input (CLKIN) pin 7-46, 7-95, 11-8, 11-10, 11-14, 11-16
Clock jitter 11-37
Clock Output (CLKOUT) pin 11-8, 11-36
Clock ratio 11-15, 11-16, 11-17
Clock ratio Configuration (CLK_CFGx) pins 7-126, 11-8, 11-15, 11-16, 11-17, A-52
Clock Rising Edge (CKRE) bit 9-10, A-72, A-74
Cluster multiprocessing 7-97, 7-98, G-2
COM port, McBSP (*See Link Ports*)
COMHALT bit 10-12
Companding (compressing/expanding) 1-15, 9-18, 9-19, 9-20, 9-28, 9-33, G-3
Compare Accumulation (CACCx) bits 2-10, A-12
Compare function 2-8
Compiler 1-18
Complementary conditions 3-59
Complementary registers 2-38, G-6
Computational instructions 2-1
Computational mode 2-39
setting 2-2
Computational status, using 2-7
Computational units (*See Processing Elements*)
Computational units, dual 2-37
Condition code Select (CSEL) bits 3-57, 7-103, A-4
Condition codes 3-55
Conditional
Branches 3-16, 3-17, 3-59, G-3
Complementary conditions 3-59
Compute operations 3-58
Conditions list 3-55
Data moves 3-59
Execution summary 3-58
Instructions 2-7, 2-39, 2-41, 3-4,

3-54, 7-5, 7-10
SIMD mode and conditionals 3-57
Conditional branches 3-17
Conditional data moves 3-59
Conditional sequencing 3-54
Conflict resolution ratio 7-3, G-3
Connections
DMA interface 6-95
External port 7-1
Host asynchronous interface 7-51
Link port 8-2
SBSRAM interface 7-45
Serial port 9-4
Constant, link port (CNSTx) registers A-71
Context switch 1-10, 2-31
Conventions 1-25
Core Instruction Fetch ($\overline{\text{CIF}}$) pin 7-7, 7-8, 7-50, 7-105, 11-3, 11-11
Core-memory halt (COMHALT) bit 10-12
Core-to-CLKIN Ratio (CRAT) bits 7-126, A-52
Count (Cx) registers 6-5, 6-9, 6-84, G-5
Counter-based loops 3-25
(*See also Non-counter-based loops*)
Crosstalk 11-44
Current Bus Master (CRBMx) bits 7-102, 7-104, 7-125, A-51
Current Loop Counter (CURLCNTR) register 3-31, A-31
Customer support 1-24

D

D unit (*See DAGs or ALU*)
DADDR register 3-7
Data (Dreg) registers G-3
data (R0-R15, S0-S15) registers A-15
Data access
(*See also Data moves*)
Conflicts 5-4
Dual-data access restrictions 5-3
Dual-data accesses 5-3
Options 5-45
Settings 5-28
Data Address Generators (DAGs) 1-9, 4-1, 4-3, 5-6, 5-10, 5-34, G-3
Data alignment 4-20
Data move restrictions 4-22
Data moves 4-19, 4-21
Enhancements 1-19
Features 1-6
Instructions 4-23
Operations 4-9
Setting Modes 4-3
SIMD mode 4-19
Status 4-9
Data addressing mode 2-39

Data alignment 5-7, 5-22, 5-23, 5-41, 7-2, 7-70
External port 11-10
link data 8-10
Serial data 9-17, 9-20
Data Buffers 6-11
Data bus (DATA63-0) pins 7-3, 7-5, 7-46, 7-95, 11-3, 11-11
Data buses 1-3
Data fetch, external port 7-3, G-3
Data flow 1-6, 2-1
Data flow multiprocessing 7-97, G-3
Data format 2-2
External data 6-21
Link data 6-48
Serial data 6-52, 9-17, 9-19
Data formats
Extended precision normal word, 40-bit floating-point 2-4
Normal word, 32-bit fixed-point 2-5
Normal word, 32-bit floating-point 2-3
Short word, 16-bit floating-point 2-5
Data Hold Cycle 7-13
Data Independent Receive Frame Synch (DIRFS) bit 9-10
Data Independent Transmit Frame Synch (DITFS) bit 9-9, 9-11, 9-27, A-72
Data Memory (DM) bus 1-3
Data memory address (DMAx) register 10-15
Data memory data select (DMDSEL) bit 10-10
Data moves 1-12, 7-74
40/48-bit 7-117
Conditional 3-59
External memory limits 7-9
Fixed sizes 5-11
Instructions 7-69
Limits 7-9
Moves to/from PX 5-12
Serial port 9-36
Data packing 1-13, 7-4, 7-75, 7-117, 8-12, 9-18, 9-37, 9-38
Data Receive (DRx) pin 9-3, 11-7, 11-11
Data registers 1-7, 2-29, 2-39, G-3
Data registers, secondary hi/lo (SRRFH/L) bits 2-32
Data sampling, serial 9-25
Data Transmit (DTx) pin 9-3, 9-30, 11-7, 11-11
Data Type, external port (DTYPE) bit 6-15, 6-18, 6-21, A-55
Data Type, serial port (DTYPE) bits 6-50, 9-19, 9-20, A-71, A-74
Data types 5-40
Data, fixed- and floating-point 2-1, G-1
Data, serial framed and unframed 9-24
Deadlock (*See Bus deadlock*)
Deadlock resolution G-4

Debugging
Tools 11-24
Decode Address (DADDR) register A-30
Decode address register 3-3
Decode cycle 3-9
Decoupling capacitors 11-45
Decrement instruction 2-8
def21160.h file A-82
Delayed branch (DB) Jump or Call 3-16, 3-17, 3-19, 3-20, G-4
Denormal operands 2-4
Deposit bit field 2-21
Development tools 1-16
device identification register 10-37
Differences between SHARCs 1-19
Dimension A and B (DAx and DBx) registers 6-6, 6-10, 6-84, G-5
Diodes, protection 11-22
Direct branch 3-16, 3-17, G-4
Direct reads and writes 7-59, 7-68, 7-70, 7-72, 7-75, 7-116, 7-118, G-4
Direct write 7-70
Asynchronous interface 7-71
FIFO 6-59, 7-70, 7-117
Latency 7-71, 7-118
Direct Write Pending (DWPD) bit 7-72, 7-83, 7-123, 7-125, A-51
DM/PM address E, M, and S fields A-29
DMA
Boot memory DMA 6-16
Bus slave versus bus master 7-75
External port 6-75, 7-74
Interrupt-driven DMA 6-55
Link port 6-81
Serial port 6-91, 9-36
Transfers 7-74
Two-dimensional 6-93
Two-dimensional, link port 6-83
DMA (Direct Memory Accessing) G-4
DMA address generator 6-8
DMA chaining G-4
DMA channel
Channels, parameters, and buffers 6-11
Interrupt priorities 6-56
Latency 6-55
Priority 6-10, 6-19, 6-45, 6-52, 6-66, 6-68, 6-69
Status 6-54
DMA Channel Priority Rotation, external port (DCPR) bit 6-15, 6-18, A-47
DMA channel Status (DMASTAT) register 6-54, A-60
DMA channels 1-16, 6-11
DMA Control (DMACx) registers 6-4, 7-110, A-54, A-55, G-5
DMA controller 1-3, 1-15
Enhancements 1-21, 1-22
Operation 6-65
Priority pathways 6-68
DMA data
16-bit external transfers 6-27
32-bit external transfers 6-27

32-bit internal transfers 6-29
48-bit internal transfers 6-29
64-bit external burst transfers 6-26
64-bit external transfers 6-25
64-bit internal transfers 6-28
DMA Enable, external port (DEN) bit 6-13, 6-15, A-55
DMA external request counter 6-37
DMA Grant (DMAGx) pins 6-34, 6-41, 6-96, 7-75, 7-105, 11-6, 11-11
DMA hardware handshake 6-34, 6-39, 6-96
DMA hardware interface 6-95
DMA hold off 6-32, 6-38
DMA Internal Request and Grant Paths 6-68
DMA parameter registers G-5
DMA pipeline 6-37
DMA Request (DMARx) pins 6-30, 6-33, 6-34, 6-40, 6-96, 7-75, 11-6, 11-11
DMA sequences
Chain insertion 6-73
Chain set up and start 6-73
Chaining sequences 6-70
Sequence complete interrupt 6-55
Sequence end 6-66
Sequence start 6-65
TCB loading 6-71, G-5
DMA targets
External memory 6-23
Internal memory 6-93
DMAx register 10-15
DMDSEL bit 10-10
DMxx register 10-13, 10-15
Do Until instruction 3-23 (*See also Loops*)
DRAM
Page fault 7-17
Page size 5-38
Paged-mode 1-13
DSP
Architecture overview 1-7
Defined 1-1
Design advantages 1-2
Dual add and subtract 2-34
Dual processing element moves (*See Broadcast load*)
Dual-data accesses 5-45, 5-46

E

E field, address A-28
Edge-sensitive interrupts 3-40, A-7, G-5
Effect latency (*See Latency*)
EIRQENA bit 10-8
ELAST (address last) page register A-54
E-mail for information 1-24
Emulation (JTAG) 1-3
Emulation status EMU pin 11-28
emulator 48-bit PX shift (EMUPX) register 10-6
emulator 64-bit PX shift (EMU64PX) register 10-7

emulator control shift (EMUCTL) register 10-7
emulator enable (EMUENA) bit 10-8
emulator idle (EMUIDLE) instruction 10-17
Emulator interface for ADI JTAG DSPs
  Illustrated 11-30
emulator interrupt (EMUI) bit A-18
emulator interrupt enable (EIRQENA) bit 10-8
emulator Nth event counter (EMUN) register 10-16
emulator PC shift (EMUPC) register 10-7
emulator PM data shift (EMUPMD) register 10-5
Emulator pod
  connection 11-32
emulator ready (EMUREADY) bit 10-12
emulator space (EMUSPACE) bit 10-12
emulator status ($\overline{\text{EMU}}$) pin 11-9
emulator status shift (EMUSTAT) register 10-11
enable breakpoint (ENBx) bit 10-10
enable $\overline{\text{BRKOUT}}$ pin 10-8
ENBx bit 10-10
Endian Format, Little Versus Big G-5
End-of-loop 3-24
EPAx register 10-16
EPHALT bit 10-12
EPROM Boot (EBOOT) pin 11-7
EPROM booting 6-77, 6-78
EPSTOP bit 10-9
EPx register 10-13, 10-15
Equals (EQ) condition 3-55
ESD protection 11-22
Execute cycle 3-9
Executing instructions from external memory 7-50
Execution stalls, bus transition 7-105
Explicit versus implicit operations G-6
Exponent derivation 2-1, G-13
Extended precision normal word 5-22, 5-43
  Broadcast load 5-74
  Data access 5-62
  Data move limits 7-9
  Data storage 5-1
  Mixed data access 5-43
  SIMD mode access 5-64
  SISD mode access 5-64
External Bank Access Mode (EBxAM) bits 5-29, 5-36, A-49
External Bank x Waitstates (EBxWS) bits 5-29, 5-37, 7-26, A-49, G-15
External bus G-6
External bus arbitration 6-69
External Bus Priority (EBPRx) bits 5-29, A-47

External DRAM Last address (ELAST) register 7-16
External handshake mode (EXTERN) bit 6-16, 6-22, 6-24, 6-29, 6-30, 6-33, 6-40, A-57
External memory 1-21, 5-12, 5-20, G-6
  Access modes 5-36, G-10
  Access timing 7-18
  Banks 7-10
  Data moves 5-11
  Direct write limits 7-9
  Instruction fetch 7-49
  Interface 7-3
  SIMD mode accesses 7-9
external memory addresses A-29
External memory DMA Count (ECx) registers 6-6, 6-10, 6-30, 6-40, A-62, G-5
External memory DMA Index (EIx) registers 6-6, 6-10, 6-30, 6-40, A-61, G-5
External memory DMA Modifier (EMx) registers 6-6, 6-10, 6-30, 6-40, A-61, G-5
External port 1-3, 1-13, 5-4, 7-1, G-6
  Buffer modes 6-17
  Buffer status 6-58
  Conflict resolution 7-3, G-3
  Data alignment 11-10
  Data packing 1-13
  DMA channel priority modes 6-18
  DMA channel priority swap 6-69
  DMA channel transfer modes 6-21
  DMA handshake modes 6-22
  DMA sequences 6-75
  DMA setup 6-75
  Enhancements 1-21
  Latency G-3
  Modes 6-13, 7-2
  Packing status 6-58
  Read/write timing 7-105
  Status 6-57
  Termination values 11-11
External port address (EPAx) register 10-16
External port Boot (EBOOT) pin 6-77, 6-89
External Port Buffer (EPBx) registers 6-4, 7-121, A-52
External Port buffer x DMA interrupt (EPxI) bit A-20, A-21
External port Bus Priority (PRIO) bit 6-16
External port DMA
  Channels 6-23
  DMA hardware interface 6-95
  DMA setup 6-75
  Modes 6-21, 6-22
External port FIFO buffers 6-59, G-6
External port halt (EPHALT) bit 10-12

External port Packing Mode (PMODE) bits 6-15
External port stop (EPSTOP) bit 10-9
External-Handshake mode 6-23, 6-40
  Program Control (PCI) interrupt 6-41
  Transfer Size 6-42
Extract bit field 2-21
Extract exponent 2-21

F

FADDR register 3-7
False always (FOREVER) Do/Until condition 3-56
FAX for information 1-23
Fetch Address (FADDR) register 3-3, A-30
Fetch cycle 3-9
Fetched address 3-3
Fetching instructions from external memory 7-50
Field deposition/extraction 2-1, 2-23, G-13
FIFO buffer Status, external port (FS) bits 6-58, A-58
File Transfer Protocol (FTP) site 1-23
Fixed priority 6-19, 6-45, 6-52, 6-69, 7-99
Fixed-point
  ALU instructions 2-11
  Data 2-1, G-1
  Multiplier instructions 2-19, 2-35
  Operands 2-9
  Operations 2-30
  Saturation values 2-17
Fixed-point Overflow Interrupt (FIXI) bit 3-43, A-21
Flag input (FLAGx_IN) conditions 3-56
Flag input/output (FLAGx) pins 7-87, 11-5, 11-18
Flag input/output select (FLGxO) bits 11-19, A-7
Flag input/output value (FLAGS) register 11-19, A-27
Flag update 2-10, 2-18, 2-26, 2-43, 3-43, 4-9, 5-39, 7-87, G-7
FLAGx input/output value (FLGx) bits A-27
Floating-point
  ALU instructions 2-12
  Data 2-1, 2-6, G-1
  Data format (RND32) bit 2-2
  Multiplier instructions 2-20
  Operations 2-30, 2-35
Floating-Point DSP (Why?) 1-2
Floating-point Invalid Interrupt (FLTII) bit 3-43, A-22
Floating-point Overflow Interrupt (FLTOI) bit 3-43, A-22
Floating-point Underflow Interrupt (FLTUI) bit 3-43, A-9, A-22
Flow-through SBSRAM (*See SBSRAM*)

Flush DMA buffers/status (FLSH) bit 6-13, 6-58, A-57
Format (*See Data format*)
Format conversion 2-8
Format packing (Fpack/Funpack) instructions 2-5
Fractional
  Data 2-5, 2-6
  Input(s) 2-20
  Results 2-15, C-6
Frame synch
  Active low 9-24
  Early versus late 9-25
  Frequencies 9-3
  Internal versus external 9-24
  Sampling 9-25

G

General Purpose (GPx) registers 6-6, 6-9, 6-71, 6-87, G-5
General Purpose DMA (GPx, DBx, DAx) registers A-60
global interrupt enable A-4
Greater or Equals (GE) condition 3-55
Greater Than (GT) condition 3-55
Ground plane 11-44

H

Handshake and Idle for DMA Enable (HIDMA) bit 6-15, A-50
Handshake mode 6-22, 6-33, 6-98
  Enable/disable transition 6-38
  Operation 6-35
  Transfer Size 6-35
Handshake mode (HSHAKE) bit 6-16, 6-22, 6-24, 6-29, 6-30, 6-33, 6-40, A-56
Handshaking
  External port 7-3, 7-53, 7-94
  External port DMA 6-22
  link port 8-2, 8-8
  link port timing 8-10
  Serial port 9-13
Harvard architecture 5-2, G-7
Hold off
  DRAM page fault 7-17
  DSP, bus transition 7-105
  DSP, during DMA 6-38
  External device, during DMA 6-32
  SBSRAM 7-49
Hold time cycle 5-37, 7-13, 7-14, G-7
Hold time, inputs 11-18
Host Bus Grant ($\overline{\text{HBG}}$) pin 7-53, 7-54, 7-84, 7-95, 7-105, 7-114, 11-5
Host Bus Master (HSTM) bit 7-125, A-51
Host Bus Request ($\overline{\text{HBR}}$) pin 7-53, 7-54, 7-122, 11-5
Host interface 1-14
  access to link buffers 8-12
  Booting 6-77
  Enhancements 1-21
  Multiple DSP interface 7-92

Read and write timing 7-62
Single DSP interface 7-51
System bus hardware 7-84
Uniprocessor 7-60
Host Most Significant Word First packing (HMSWF) bit 6-14, 7-80, A-45
Host Packing Mode (HPM) bits 6-14, 6-17, 7-55, 8-12, A-45
Host Packing Status (HPS) bits 6-57, 7-126, A-52
Host Packing status Flush (HPFLSH) bit 6-57, 7-79, A-46
Host Transition Cycle (HTC) 7-55, 7-65, 11-11, G-7
Hypertext links 1-26
Hysteresis on Reset ($\overline{\text{RESET}}$) pin 11-36

I

I/O address (IOAx) register 10-15
I/O interrupt conditions 6-53
I/O processor 1-3, 1-14, 5-14, 6-1, 6-3, 6-6, 7-3
Data moves 5-11
DMA channel priority 6-68
External port modes 6-13
Link port modes 6-41
Registers A-33, G-8
Serial port modes 6-49
Shadow registers 7-68, 7-117
Status 6-53
I/O stop (IOSTOP) bit 10-8
ICSA pin 10-2
ID Code (IDC) bits 7-125, A-51
Identity, DSP (ID2-0) pins 7-101, 7-102, 11-6, 11-11
Idle cycle 7-11, 7-14, G-8
Idle instruction 3-2, 3-48, G-8
IEEE 1149.1 JTAG specification 1-16, 10-1, 10-4, 10-38, G-9
IEEE 1149.1 JTAG standard 11-31
IEEE 754/854 floating-point data format 2-4, C-1
IEEE Floating-point (*See Extended precision normal word*)
IEEE floating-point number conversion 2-5
Illegal I/O processor Register Access (IIRA) bit A-14
Illegal I/O processor register Access Enable (IIRAE) bit 5-29, 5-35, A-8
Illegal Input Condition Detected (IICD) bit 5-35, 5-36, A-18
Immediate branch 3-18
Implicit operations 5-35
Broadcast load 4-6
Complementary registers 2-38
Long Word (LW) accesses 5-41
Neighbor registers 5-42
SIMD mode 2-38
IN idle mode (INIDLE) bit 10-12
In-circuit signal analyzer (ICSA) function 10-10
in-circuit signal analyzer (ICSA) function 10-17

Increment instruction 2-8
Index (Ix) registers 4-3, 4-16, A-32, G-8
Indirect addressing 1-10
Indirect branch 3-16, 3-17, G-8
Inductance (run length) 11-41
Inexact flags 2-4
Infinity, round-to 2-4
INIDLE bit 10-12
Input filtering, link port 11-35
Input Mask, keyword comparison (IMAT) bit 9-33
Input Mode, keyword comparison (IMODE) bit 9-33
Input setup and hold time 11-18
Input signal conditioning 11-35
Input/Output (IO) bus 1-3
Instruction
  External memory fetch 7-3, 7-7, 7-49, 7-69, G-3
  Moves 7-69, 7-117
Instruction (bit) 3-6
Instruction cache 1-11, 3-10, 5-3
Instruction cache architecture
  Illustrated 3-11
Instruction dispatch/decode (*See Program Sequencer*)
Instruction pipeline 3-3, 3-9
Instruction register 10-4
Instruction set 1-1, 1-25
  Changes 1-22
  Enhancements 1-22
Instruction word
  Data access 5-43
  Instruction moves 5-11
  Storage 5-1
  Word Rotations 5-23
Instruction Word Transfer (IWT) bit 7-60
instructions
  EMUIDLE 10-17
Integer
  Input(s) 2-20
  Results 2-15, C-7
Integer data 2-5
Interleaved data 5-84, G-8
internal (system) memory addresses A-29
Internal Buses 1-11
Internal I/O Bus Arbitration (Request & Grant) 6-66
Internal Interrupt Vector Table (IIVT) bit 5-28, 5-32, A-45
Internal memory 5-1, 5-14, 5-21, G-9
  Shadow write FIFO 7-73
Internal Memory 32-bit transfers (INT32) bit A-58
Internal Memory Data Width (IMDWx) bits 5-9, 5-28, 5-32, 5-39, 6-21, 6-48, A-46
Internal memory DMA Count (Cx) registers A-59
Internal memory DMA Index (IIx) registers 6-5, 6-6, 6-84, A-58, A-59, G-5

Internal memory DMA Modifier (IMx) registers 6-5, 6-6, 6-84, A-59, G-5
Internal Receive Frame Synch (IRFS) bit 9-10, 9-24, A-74
Internal serial Clock (ICLK) bit 9-10, A-72, A-74
Internal Transmit Frame Synch (ITFS) bit 9-11, 9-24, 9-30, A-72
Interrupt controller 3-3
Interrupt Enable, global (IRPTEN) bit 3-42, A-4
Interrupt input ($\overline{\text{IRQ2-0}}$) pins 11-5, 11-18
Interrupt input x interrupt (IRQxI) bit A-19, A-20
Interrupt Latch (IRPTL) register A-18
Interrupt latency 3-35, 9-5
  Cache miss 3-36
  Delayed branch 3-36
  $\overline{\text{IRQ}}$x and multiprocessor vector standard 3-38
  Single-cycle instruction 3-36
  Writes to IRPTL 3-35
Interrupt Mask (IMASK) register 3-42, 9-38, A-22
Interrupt Mask Pointer (IMASKP) register 3-46, A-23
Interrupt mask/mask pointer, link port (LIRPTL) register 3-42, 3-46
Interrupt nesting enable (NESTM) bit 3-46
Interrupt x Edge/level sensitivity ($\overline{\text{IRQ}}$xE) bits 3-41, A-7
Interrupt-driven I/O, external port (INTIO) bit 6-18, 6-57, 6-60, A-56
Interrupt-driven transfers
  External port 6-60
  Link port 6-63
  Serial port 6-65
Interrupting IDLE 3-48
Interrupts 1-10, 2-7, 3-2, 3-33, 4-9, 5-35, 5-36, 5-39, A-18, A-19, G-9
  Arithmetic 3-43
  Clear interrupt (CI) Jump 3-48
  Data Address Generators (DAGs) 4-15
  Delayed branch 3-20
  DMA interrupts 6-55, 6-56, 6-60
  Hold off 3-39
  Idle instructions 3-48
  Inputs ($\overline{\text{IRQ2-0}}$) 3-34
  Interrupt nesting A-4, A-23
  Interrupt sensitivity 3-40, A-7, G-9
  Interrupt vector table 5-31, B-1
  IRPTL write timing 3-35
  Latency (*See Interrupt latency*)
  link ports 8-12
  Masking 3-41
  Masking and latching 3-41, 3-42, 6-56, 8-14

Multiprocessing 3-49
PC stack full 3-53
Program Control (PCI) interrupts 6-41
Response 3-34
Re-using 3-47
Serial port 9-5, 9-38
Software 3-35
spurious, link port 8-7, 8-16
Timer 3-51
Vector 3-49
Vector interrupts 7-81, G-11
Interrupts and sequencing 3-33
Interval timer 3-50
IO architecture 1-22
IO pin, ESD protection 11-22
IOAx register 10-15
IOSTOP bit 10-8
IRPTEN bit 3-42
IRQ0E bit 3-41
IRQ1E bit 3-41
IRQ2-0 3-45
IRQ2E bit 3-41

J

JTAG boundary register 10-18
JTAG emulator references 11-31
Additional documents 11-31
JTAG instruction register codes 10-4
JTAG port 1-3, 1-16, 10-1, 10-3, 11-24, G-9
JTAG scan chain
Restrictions 11-29
JTAG signals 11-30
Listed 11-27
JTAG target board connector with no local boundary 11-27
JTAG test access port (TAP) 10-3, 11-24, 11-30
JTAG test access port (TAP) pin 10-3
Jump instructions 3-1, 3-15, G-9
Clear interrupt (CI) 3-16, 3-48
Conditional 3-16
Delayed branch 3-16
Loop abort (LA) 3-16, 3-22
Pops status stack with (CI) 3-45

K

Keyword receive comparison (KEYWDx) registers 9-7, 9-33, A-81
Keyword receive comparison Mask (KEYMASKx) registers 9-7, 9-33, A-81

L

L unit (*See ALU*)
LADDER register 3-7
Latchup 11-35
Late Frame Synch (LAFS) bit 9-11, 9-12, A-72, A-75
Latency 3-6, 3-13, 3-35, 6-49, 6-55, 6-87, A-33, G-3
Direct read 7-72
Direct write 7-71
Direct write pending 7-83

DMA status A-61
Input Synchronization 11-17
Instruction fetch, external memory 7-3
link ports 8-11
Serial port interrupts 9-5
Serial port registers 9-12
Shadow registers 7-68
Slave write FIFO 7-71
Synchronous write 7-28
System registers 3-6
Vector interrupt 3-38
LCTLx register 8-7
Least significant bits (LSB) 3-11
Length (Lx) registers 4-3, 4-17, A-33, G-9
Less or Equals (LE) condition 3-55
Less than (LT) condition 3-55
Level-sensitive interrupts 3-40, A-7, G-9
Line run length (inductance) 11-41
Line termination 11-40
Link port 8-21
Serial port 9-38
Link buffer DMA Chaining Enable (LxCHEN) bit 6-43, 6-47, A-63
Link Buffer DMA Enable (LxDEN) bit 6-13, 6-43, 6-47, 8-11, A-63
Link buffer Enable (LxEN) bit 6-43, 6-44, A-63
Link Buffer Extended word size (LxEXT) bit 6-44, 6-48, 8-11, A-64
Link Buffer Mesh Signal Processing (LMSP) bit 8-6, A-66
Link Buffer Receive Error, pack status (LRERRx) bits 6-60, 8-16, A-66
Link Buffer Status (LxSTATx) bits 6-60, 8-10, A-65
Link buffer Transmit/receive (LxTRAN) bit 6-43, 6-48, A-63
Link Buffer Two-Dimensional DMA enable (LxDMA2D) bit 6-44, 6-48, 6-56, 6-84, A-64
Link Buffer x DMA interrupt mask (LPxMSK) bits A-25
Link Buffer x DMA interrupt mask pointer (LPxMSKP) bits A-26
Link buffer-to-port assignment 6-44
Link Path Delay, mesh multiprocessing (LPATHD) bits 8-6, A-66
Link port 1-3, 1-15, 1-22, 6-47, G-9, G-10
Booting 6-88, 6-89
Buffers 6-44, 6-62, A-67
Channel rotating priority 6-46
Data transfers, cluster 7-100
DMA 6-47, 6-81
Enhancements 1-22
Identifying the one to service 8-15
Interrupt-driven transfers 6-63
Line termination 8-21
Priority modes 6-45
Status 6-60, 6-61

Transmission errors 8-16
link port 8-1
booting 8-5
buffers 8-3
designing for link ports 8-21
DMA 8-5, 8-12
handshake timing 8-10
Interrupts 8-12, 8-16
interrupts with DMA 8-13
interrupts with service request 8-14
interrupts without DMA 8-14
loopback mode 8-4
pulldown resistor enabled 8-7
throughput 8-22
token passing 8-17
transmission errors 8-16
transmitted word 8-8
Link Port Acknowledge (LxACK) pins 8-2, 8-8, 8-10, 11-7, 11-11
Link port Assignment (LAR) register 6-5, 8-4, A-67
Link port Boot (LBOOT) pin 6-77, 6-89, 11-7
Link Port Buffer (LBUFx) registers 6-4, 8-3, 8-10, A-62
Link port Buffer Control (LCTLx) registers A-62, A-63
link port buffer x DMA interrupt (LPxI) bits A-24
Link port Clock (LxCLK) pins 8-2, 8-8, 8-10, 11-7, 11-11
Link port Clock Divisor (LxCLKDx) bits 8-6, 11-15, 11-16, A-64
Link port Common controls (LCOM) register 6-4, A-65
Link port Control (LCTLx) registers 6-5, 6-42
Link port Data (LxDAT7-0) pins 8-2, 8-3, 8-10, 11-7, 11-11
Link port Data Path Width (LxDPWID) bit 8-7, A-65
Link port DMA Channel Priority Rotation (LDCPR) bit 6-43, 6-45, A-47
Link port DMA interrupts Latch and Mask bits 3-43
Link Port Interrupt DMA Summary Interrupt (LPISUMI) bit 3-43, A-20
Link port Interrupt Latch/mask/maskpointer (LIRPTL) register 3-46, A-24
Link port Path (LPATHx) registers A-70
Link port Path Counter (LPCNTx) register A-70
Link Port Pulldown Resistor Disable/Enable (LxPDRDE) bit 8-6, 8-10, A-64
Link port Receive Mask (LxRM) bits 6-61, A-68
Link port Receive Request status (LxRRQ) bits 6-61, A-69

Link port Service Request (LSRQ) register 6-62, 8-14, A-68
Link port Service Request Interrupt (LSRQI) bit 6-56, 6-62, 8-7, 8-13, A-21
Link port Transmit Mask (LxTM) bits 6-61, A-68
Link port Transmit Request status (LxTRQ) bits 6-61, A-69
Linker 1-18
Little-endian format 1-15
Loader 1-18
Logical operations 2-7
Logical shifts 2-1, G-13
Long word 5-22, 5-41, 5-42
  Broadcast load 5-74
  Data access 5-7, 5-41, 5-67, G-12
  Data moves 4-21, 5-41
  Data storage 5-1
  SIMD mode 5-69
  Single Data 5-67
  SISD Mode 5-69
Long word (LW) mnemonic 5-9, 5-11, 5-16, 5-36, 5-41, 5-44, 7-117, G-12
Loop 3-1, 3-21, G-10
  Address stack 3-6, 3-29
  Conditional loops 3-21
  Counter stack 3-30
  End restrictions 3-23
  Status 3-30
  Termination 3-4, 3-23, 3-30, 3-31, 3-55, A-30
Loop abort (LA) Jump 3-16, 3-22
Loop address stack 3-29
Loop Address stack (LADDR) register A-30
Loop Counter (LCNTR) register 3-31, 3-32, A-31
Loop counter expired (LCE) condition 3-21, 3-56
Loop counter stack 3-30
Loop Stack Empty (LSEM) bit 3-31, A-14
Loop Stack Overflow (LSOV) bit 3-31, A-14
Loopback mode
  Link port 8-4
  Serial port 9-27, A-75
Loops and sequencing 3-21
Low active Receive Frame Synch (LRFS) bit 9-10, 9-24, A-75
Low active Transmit Frame Synch (LTFS) bit 9-11, 9-24, A-72
LPSUM bit 3-43
LRU bit 3-11
LSB 3-11
LSEM bit 3-31
LSOV bit 3-31
LxDPWID bit 8-7

M

M field, address A-28
M unit (*See Multiplier*)
Mailing address for information 1-24
Mantissa (floating-point operation) 2-8

Masking interrupts 3-41
Link port 8-14
Serial port 9-38
Master mode 6-22, 6-24
16-bit external transfers 6-27
32-bit external transfers 6-27
32-bit internal transfers 6-29
48-bit internal transfers 6-29
64-bit external burst transfers 6-26
64-bit external transfers 6-25
64-bit internal transfers 6-28
Controls 6-25
Internal address/transfer size generation 6-28
Transfer Size 6-25
Master mode enable (MASTER) bit 6-16, 6-22, 6-24, 6-29, 6-30, 6-33, 6-40, A-56
Max/Min function 2-8
maximum burst length (MAXBL) bits A-58
Memory 1-3, 1-12, 5-1, 5-4, 5-14, 5-21, G-9
Access priority 5-3, 5-4, 5-33, 5-74
Access types 5-34, 5-39, 5-46, 9-36, G-10
Asynchronous interface 5-36, G-10
Banks of memory 5-4, 5-21, 5-33, 7-10, 7-50, G-10
Blocks of memory 5-1, 5-3, 5-4, 5-16, 5-32, G-10
Boot memory 5-21
Columns of memory 5-6
Enhancements 1-20
mapped devices 6-10
mapped registers 5-14, A-33, A-36
Memory map 5-14
Multiprocessor broadcast write 5-19
Synchronous interface 5-36, G-10
Transition from 32-bit/48-bit data 5-27
Unbanked memory 5-21
Memory bank Size (MSIZE) bits 5-28, 5-33, A-46
Memory Select ($\overline{\text{MS3-0}}$) pins 5-21, 5-33, 7-5, 7-10, 7-46, 7-50, 7-105, 11-3, 11-11
Memory Test (MTST) bit 10-11
Memory Test Shift (MEMTST) register 10-13
Memory transfers 5-46, 5-47
16-bit (Short word) 5-47
32-bit (Normal word) 4-20, 5-55
40-bit (Extended precision normal word) 5-62
64-bit (Long word) 4-21, 5-67
Message (MSGRx) registers 7-81, 7-82, 7-123, 7-124, A-53
Microprocessor interface 7-94
Mixed 32-Bit & 48-Bit Words 5-23
Mixed 32-Bit and 48-Bit Words 5-23
Mixed instructions and data

No unused locations 5-26
One unused location 5-27
Two unused locations 5-27
Mixed word width
SIMD mode 5-74
SISD mode 5-72, 5-73
Mixing 40/48-bit and 16/32/64-bit data 5-21, 5-27
µ-law companding 1-15, 9-1, 9-19, 9-20, 9-33
Mnemonics (*See Instructions*)
Mode control 1 (MODE1) register 3-6, 3-44, A-3
Mode control 2 (MODE2) register A-6
Mode Mask (MMASK) register 3-44, 4-15, A-5
MODE2_SHDW register A-53
Modified addressing 4-1, 4-10, G-10
Modify (Mx) registers 4-3, 4-16, A-32, G-11
Modify instruction 4-15, 4-18, 4-25
Modulo addressing 1-10
Most Significant Word First, packing (MSWF) bit 6-16, 6-18, A-56
MTST bit 10-11
Multichannel buffered serial port, McBSP (*See Serial Ports*)
Multichannel mode 9-28, 9-30, G-14
Frame synchs 9-30
Multichannel Receive Channel Select (MRCSx) registers 9-7, 9-32, A-80
Multichannel Receive Compand Select (MRCCSx) registers 9-7, 9-32, A-81
Multichannel serial Enable (MCE) bit 9-11, 9-30, A-75
Multichannel serial Frame Delay (MFD) bit 9-12, 9-31, A-72
Multichannel Transmit Channel Select (MTCSx) registers 9-7, 9-32, A-79
Multichannel Transmit Compand Select (MTCCSx) registers 9-7, 9-32, A-80
Multifunction computations 2-32, G-11
Multiple DSP connection to JTAG header
illustrated 11-30
Multiple DSP systems 11-28
Multiplier 1-7, 2-1, G-11
Clear operation 2-16
Input modifiers 2-20
Instructions 2-13, 2-18
Operations 2-14, 2-17
Result (MRF/B) registers 2-14, 2-15
Rounding 2-16
Saturation 2-17
Status 2-7, 2-17, 2-18
Multiplier fixed-point Overflow Status (MOS) bit 2-18, A-13

Multiplier floating-point Invalid (MI) bit 2-17, A-11
Multiplier floating-point Invalid Status (MIS) bit 2-18, A-13
Multiplier floating-point Overflow Status (MVS) bit 2-18, A-13
Multiplier floating-point Underflow (MU) bit 2-17, A-11
Multiplier floating-point Underflow Status (MUS) bit 2-18, A-13
Multiplier Negative (MN) bit 2-17, A-10
Multiplier Overflow (MV) bit 2-17, 3-55, A-10
Multiplier Results (MRx) registers A-16
Multiplier Signed (MS) bit 3-55
Multiply—accumulator (*See Multiplier*)
Multiprocessing
  Bus arbitration 7-101
  Direct read and write 7-116
  DSP Interface 7-94
  interrupts 3-49
  Local memory 7-92
  SIMD processing 7-100
  System architectures 7-97
  System hardware 7-95
Multiprocessor
  Booting 7-116
  Booting, EPROM 6-78
  Interface 1-14, 1-21
  Memory 5-12, 5-17, 7-3, 7-14, A-29, G-11
  System G-11
  Vector interrupt 3-49, G-11

N

Nearest, round-to 2-4
Negate breakpoint (NEGx) bit 10-9
NEGx bit 10-9
Neighbor registers 5-41, 5-42, 5-67, 5-69, 5-72, G-12
Nested interrupt routines 3-4
Nesting Multiple interrupts (NESTM) bit 3-46, A-4
NO boot mode (NOBOOT) bit 10-11
Non-counter-based loops 3-27 (*See also Counter-based loops*)
Normal word 5-22, 5-43 (*See also Extended precision normal word*)
  Accesses with LW G-12
  Broadcast load 5-74
  Data access 5-43
  Data move 4-20
  Data storage 5-1
  Mixing 32-bit data and 48-bit instructions 5-23
  Multiprocessor memory 5-19
  SIMD mode 5-55, 5-57, 5-59
  SISD mode 5-59
Not Equal (NE) 3-55
Not, Logical 2-8
Not-a-number (NAN) 2-4

Number of serial Channels (NCHN) bits 9-11, 9-31, A-75

O

Open drain 7-5, 7-53, 7-101
Operands 2-4, 2-8, 2-14, 2-21, 2-29, G-3
Optimizing cache usage 3-13
Optimizing DMA Throughput 6-93
Or, Logical 2-8
Oscilloscope probes 11-47
Overflow (*See ALU, Multiplier, or Shifter*)

P

Paced Master mode 6-22, 6-29
Packed data 7-8, 7-51, 7-69
Packing
  Data 1-13, 6-10, 6-17, 6-23, 6-41, 7-54, 7-117, 8-12, 9-18, 9-37, 9-38
  Data, 32/64-bit data from host 7-75
  Data, 40/48-bit data from host 7-79
  External port status 6-58
  Link port status 6-61
  Modes 6-17, 7-54
Packing 16-bit to 32-bit Words (PACK) bit 6-50, 6-52
Packing Mode (PMODE) bits 5-30, 6-17, 7-75, 7-117, A-56
Packing mode, serial (PACK) bit 9-18, 9-37, 9-38, A-72, A-74
Packing Status, external port (PS) bits 6-57, A-55
Page boundaries 7-16
Paged DRAM boundary (PAGE) pin 7-6, 7-10, 7-15, 7-53, 7-105, 11-4, 11-11
Paged DRAM Size (PAGSZ) bits 5-29, 5-38, 7-16, A-50
Paged mode DRAM 1-13, 5-21, 5-38, 7-15
Parallel assembly code (*See Multifunction computation or SIMD operations*)
Parallel operations 2-32, G-11
Pass function 2-8
PC register 3-7
PCSTK register 3-7
PCSTKP register 3-7, 3-53
Peripherals 1-3, 1-12, 5-4, 7-3, 7-7, 7-10, 8-1, 8-21, G-9, G-12
  connecting to link ports 8-21
Pins
  $\overline{\text{BRKOUT}}$ 10-8
  $\overline{\text{BRKOUT}}$ pin 10-8
  Descriptions 11-3
  Diagram 11-2
  ICSA 10-2
  Names 1-25
  Reset states 11-12
  TAP 10-3
  TCK 10-3
  TDI 10-3

TMS 10-3
TRST 10-3
Pipelined SBSRAMs (*See SBSRAM*) 7-49
Plane, ground 11-44
PLL-based clocking 11-14
PMDAx register 10-15
Pod logic
2.5V 11-33
2.5V pod logic 11-34
JTAG Connector 11-32
Pop
Loop counter stack 3-31
Program counter (PC) stack 3-15
Status stack 3-45
Port Rotate Rotating DMA channel priority, link–external ports (PRROT) bit 6-43, 6-47, 6-69, A-47
Porting from previous SHARCs
Assembly syntax 2-30
Asynchronous access timing 7-61
Booting 6-77, 6-89
Buffer Hang Disable (BHD) bit 1-21
Bus lock 7-88
Circular Buffer Enable (CBUFEN) bit 1-20, 4-5
Conditional instructions 7-5, 7-10
Core Instruction Fetch (CIF) pin 7-50
External memory interface 7-8
In-circuit signal-analyzer (ICSA) pin 10-2
Instruction Word Transfer (IWT) bit 7-60, 7-70
link ports 8-2, 8-7
Multiprocessor Memory Space Waitstates (MMSWS) bit 7-14
Paged DRAM boundary 7-13
Performance 2-37
Program sequencer 7-9
Secondary processing element 2-37
Shadow write FIFO 7-73
shared DMA channels 8-12
Symbol changes 1-23
Synchronous access mode 7-54, 11-9
Post-modify addressing 1-10, 4-1, 4-10, 4-24, G-12
Power sequence 11-31
JTAG emulator 11-31
Precision 1-6, 2-3, 2-4, 2-5, G-12
Pre-modify addressing 1-10, 4-1, 4-10, 4-25, G-12
Primary registers 1-10, 2-29
Priority Access (PA) pin 7-95, 7-102, 7-110, 11-7, 11-11, A-57
Priority, DMA requests (*See also DMA channel priority, Rotating priority, and Fixed priority*) 6-18
Priority, external port-bus (PRIO) bit 7-110, A-57
Priority, fixed and rotating 7-99
private instructions

reserved for emulation and memory test 10-37
Probes, oscilloscope 11-47
processing element registers A-15
Processing Element Y Enable (PEYEN) bit, SIMD mode 2-3, 2-37, 4-4, 4-6, 4-19, 5-29, 5-34, A-5
Processing elements 1-3, 1-7, 1-8, 2-1, 2-30
Processor core 1-7
access to link buffers 8-11
access to serial port buffers 9-14
Buses 1-11, 1-20
Enhancements 1-19, 1-20
Program 3-1
Program Control Interrupt (PCI) bit 6-41, 6-56, 6-70, 6-71
Program Counter (PC) register 3-3, A-28
Program counter (PC) relative address 3-17, G-4
Program counter (PC) stack 3-52
Program counter (PC) stack empty (PCEM) bit 3-53
Program counter (PC) stack full (PCFL) bit 3-53
Program counter (PC) stack full (SOVFI) interrupt 3-53
Program Counter Shadow (PC_SHDW) register A-53
Program Counter Stack (PCSTK) register 3-6, A-29
Program counter stack empty (PCEM) bit A-14
Program counter stack full (PCFL) bit A-14
Program Counter Stack Pointer (PCSTKP) register 3-6, 3-53, A-29
Program fetch (*See Program Sequencer*)
Program flow 3-9
Program flow variations 3-3
Program Memory (PM) bus 1-3
Program Memory Address (PMDAx) register 10-15
Program memory bus Exchange (PX) register 1-12, 5-7, 5-32, 7-9, A-16
Program Sequence Address (PSAx) register 10-15
Program sequencer 1-3, 1-6, 1-9, 3-1, 3-3
block diagram 3-3
Latency 3-6
registers A-17
Programming information 1-1
PSAx register 10-15
PSx, DMx, IOx, and EPx (Breakpoint) register 10-13, 10-15
Pull-down resistors, link port 8-10
Push
Loop counter stack 3-31
Program counter (PC) stack 3-15
Status stack 3-44

R
Read High/Low (RDH/L) pins 6-30, 7-6, 7-46, 7-50, 7-61, 7-95, 7-105, 11-3, 11-11
Read, synchronous timing 7-64
Ready-Host Acknowledge (REDY) pin 6-32, 7-53, 7-54, 7-61, 7-65, 7-84, 7-95, 11-6
Receive Clock (RCLKx) pins 9-3, 9-21, 11-7, 11-11
Receive Clock Divisor (RCLKDIVx) bits 11-16, A-78
Receive comparison (*See Keyword and Keymask registers*)
Receive Count (RCNTx) registers A-79
Receive data (RXx) registers 6-4, 9-7, 9-12, 9-13, 9-17, 9-20, A-76
Receive Data buffer Status (RXS) bits 6-64, 9-38, A-76
Receive Divisors (RDIVx) registers 9-3, 9-7, 9-14, 11-15, A-78
Receive Frame Synch (RFSx) pins 9-3, 9-22, 9-30, 11-7
Receive Frame Synch Divisor (RFSDIV) bits 9-15, A-79
Receive Frame Synch Required (RFSR) bit 9-10, 9-22, A-74
Receive Overflow status (ROVF) bit 6-64, 9-13, A-76
Receive sign extension 9-20
Reciprocal function 2-8
references 10-38
Reflective semaphores 7-73, 7-99, 7-118
register codes
  JTAG instruction 10-4
Register files 2-29, G-3
  (*See Data register files*)
  Write precedence 2-29
register groups (I/O Processor) A-34
Register handshake/write-back messaging 7-82
Register load broadcasting (*See Broadcast load*)
Register names 1-25
Registers
  boundary 10-17
  BRKSTAT 10-12, 10-13
  DADDR 3-7
  Decode address 3-3
  device identification 10-37
  DMAx 10-15
  DMxx 10-13, 10-15
  EMU64PX 10-7
  EMUCLK 10-16
  EMUCLK2 10-16
  EMUCTL 10-7
  EMUN 10-16
  EMUPC 10-7
  EMUPMD 10-5
  EMUPX 10-6
  EMUSTAT 10-11
  EPAx 10-16
  EPx 10-13, 10-15
  FADDR 3-7
  Fetch address 3-3

instruction 10-4
IOAx 10-15
JTAG boundary 10-18
LADDER 3-7
LCTLx 8-7
MEMTST 10-13
MMASK 3-44
MODE1 3-6, 3-44
PC 3-7
PCSTK 3-7
PCSTKP 3-7, 3-53
PMDAx 10-15
PSAx 10-15
PSx 10-13, 10-15
Status 3-4
STKYx 3-53
TCOUNT 3-50, 3-52
TPERIOD 3-50, 3-51, 3-52
Registers, complementary (*See Complementary registers*)
Registers, neighbor (*See Neighbor registers*)
Register-to-register
Moves 2-42, 5-8
Swaps 2-41, 2-42, G-13
Transfers 2-40, 3-59
Related documents 1-24
Reset ($\overline{\text{RESET}}$) pin 7-95, 11-8, 11-36
Reset interrupt (RSTI) bit A-18
Resistors, Pull-up/down 11-28
Restrictions on ending loops 3-23
Restrictions on short loops 3-24
Results (MRF/MRB) registers 2-31
Return (RTI/RTS) instructions 3-15, 3-35
Re-using interrupts 3-47
Rotate (*See Swap operator*)
Rotate bits 2-21
Rotating priority 6-19, 6-45, 6-69, 7-99
Rotating Priority Bus Arbitration (RPBA) pin 7-102, 7-107, 11-6
Rounded output 2-20
Rounding 32-bit data (RND32) bit A-4
Rounding mode 2-2, 2-6, A-4
Round-to-infinity 2-4
Round-to-nearest 2-4, 2-6
Round-to-zero 2-4, 2-6
RS-232 ports 9-5

S

S field, address A-28
S unit (*See Shifter*)
Saturation (ALU saturation mode) G-13
Saturation maximum values 2-17
SBSRAM
DSP pins 7-45
Hold off 7-49
Support 7-48
Scale (floating-point operation) 2-8
Secondary processing element 2-36, 2-37
Secondary registers 1-10, 2-31, 2-40, 4-4, 4-6, 4-7, A-3

Secondary Registers for Computational Units (SRCU) bit 2-32, A-3
Secondary Registers for DAGs (SRDxH/L) bits 4-4, A-3
Secondary Registers for Register File (SRRFH/L) bit A-3, A-4
Semaphores 7-73, 7-121, 7-124, G-13
Sensing interrupts 3-40
sensitivity, interrupts A-7
Sequencing instructions from external memory 7-50
Sequential bursts, external port 7-74
Serial port (SPORT) 1-15, 9-1, 9-5, G-13
  Buffers 6-51, 6-64
  Clock frequency 9-15
  Data buffering 9-13
  Data formats 9-17
  Data/synch sampling 9-25
  DMA 6-52, 6-91, 9-18, 9-36
  Frame synch 9-16, 9-24, 9-25
  Framed versus unframed data 9-24
  Internal memory access 9-36
  Interrupt latency 9-5
  Interrupt-driven transfers 6-65
  Latency, registers 9-12
  Loopback mode 9-27
  Multichannel operation 9-28, 9-30, G-14
  Priority modes 6-52
  Reset 9-6
  Status 6-63
  Transfer modes 6-52
Serial port Chained DMA enable (SCHEN) bit 6-51, 6-52, A-72, A-75
Serial port DMA Enable (SDEN) bit 6-13, 6-50, 6-52, A-72, A-75
Serial Port Enable (SPEN) bit 6-50, 6-65, 9-6, A-71, A-74
Serial Port Loopback (SPL) bit 9-11, 9-27, A-75
Serial port Path (SPATHx) registers 9-7, A-81
Serial Port Path Count (SPCNTx) registers A-82
Serial port Receive Control (SRCTLx) registers 6-5, 6-49, 9-7, 9-9, A-73
Serial port Receive x Interrupt (SPRxI) bit 9-5, A-20
Serial port Transmit Control (STCTLx) registers 6-5, 6-49, 9-7, 9-9, A-71
Serial port Transmit x Interrupt (SPTxI) bits 9-5, A-20
serial scan paths 10-5
Serial word Endian (SENDN) bit 6-50, 6-51, A-71, A-74
Serial word Length (SLEN) bits 6-50, 6-51, 9-16, 9-17, A-71, A-74
Set, bit 2-21
Setup time, inputs 11-18

Shadow write FIFO 5-21, 7-73, 7-120
SHARC G-13
(*See also Porting from previous SHARCs*)
Background information 1-19
Defined 1-1
Shift bits 2-21
Shifter 1-7, 2-1, 2-21, 5-31, G-13
Instructions 2-27
Operations 2-21, 2-26
Status flags 2-26
Shifter input Sign (SS) bit A-12
Shifter Overflow (SV) bit 3-55, A-11
Shifter Zero (SZ) bit 3-55, A-12
Short (16-bit data) Sign Extend (SSE) bit 2-3, 5-44, A-4
Short word 5-22, 5-44
Broadcast load 5-74
Data access 5-44
Data storage 5-1
SIMD mode 5-48, 5-53, 5-57
SISD mode 5-47, 5-48, 5-51
sign extension A-4
Signal skew
Minimizing skew 11-28
Signed
data 2-5
input 2-20
SIMD mode 3-54, 5-44, 5-46, A-5
and sequencing 3-57
Complementary registers 2-38
Computational operations 2-40
Defined 2-36
Status flags 2-43
SIMD multiprocessing 7-100
Single-data access 5-46
Single-Step (SS) bit 10-8
Single-word transfers G-14
External port 7-74
Serial port 9-36, 9-37
SISD mode 5-44, 5-46
Defined 1-8
Unidirectional register transfer 2-42
Slave Direct Reads and Writes (*See Direct Read and Direct Write*)
Slave mode 6-22, 6-30, 6-97, G-2
Operation 6-31
Transfer size 6-34
Slave write FIFO 7-60
Software interrupt x, user (SFTxI) bit A-22
software interrupt x, user (SFTxI) bit A-22
Software Reset (SRSTI) bit A-45
Software Reset, emulation (SYSRST) bit 10-8
Software UARTs 9-5
spurious interrupts, link port 8-7, 8-16
square root, reciprocal 2-8
SRAM (memory) 1-3
SS bits 10-8
SSEM bit 3-44
SSOV bit 3-44

Stack Overflow/Full (SOVFI) bit A-19
Stacking status during interrupts 3-44
Stacks and sequencing 3-52
Status 5-39
  Host interface 7-81
  Link port 8-10
  Serial port 9-14
Status registers 3-4
Status stack 3-44
  Pop 3-45
  Push 3-44
Status Stack Empty (SSEM) bit 3-44, A-14
Status Stack Overflow (SSOV) bit 3-44, A-14
STATx bit 10-13
Sticky Status (STKYx/y) registers 2-7, 2-18, 3-53, A-8
STKYy bit 3-43
Subroutines 3-1, G-14
Subtract instructions 2-35
Subtract with borrow 2-8
Subtract/Add 2-8
Subtract/multiply 2-1, G-11
Support (technical or customer) 1-24
Suspend Bus Three-state ($\overline{\text{SBTS}}$) pin 7-17, 7-53, 7-87, 7-89, 7-95, 11-5
Swap register operator 2-41, G-13
Synchronous access mode 5-36, 7-5, 7-6, 7-11, 7-14, 7-18, 7-53, 7-101, 11-10, G-10
  Broadcast writes 7-65
  Deadlock resolution 7-88
  Direct read 7-72
  ID=0 unsupported 7-65
  Interface timing 7-23
  Read/Write—bus slave 7-23
  Read—bus master 7-25
  Write, One Waitstate Mode 7-32
  Write, Zero Waitstate Mode 7-28
Synchronous burst access mode
  Interface timing 7-34
  Length determination 7-36
  Read transfers 7-67
  Read/Write—bus slave 7-34
  Reads—bus master 7-38
  Stall Criteria 7-36
  Writes—bus master 7-40
Synchronous burst read, external port buffers 7-121
Synchronous Burst Static RAM (*See SBSRAM*)
Synchronous output clock (CLKOUT) pin 7-5
Synchronous transfers 7-54, 7-64, G-14
system (Sreg) registers A-2
  program sequencer A-17
System bus interfacing 7-83
System Configuration (SYSCON) register 6-4, A-45
System control registers 3-6

System design
Layout requirements 11-30
Pod specifications 11-32
Routing signals 11-30
System Status (SYSTAT) register 7-81, A-51

T

TAP pin 10-3
Target board connector 11-25
For emulator probe 11-25
TCB chain loading 6-69, 6-70, 6-71, G-14
TCK pin 10-3
TCOUNT register 3-50, 3-52
Technical support 1-24
Termination codes
(*See Condition codes and Loop termination*)
Termination values 11-40
Link port 8-21
Serial port 9-38
test access port (TAP) (*See JTAG port*)
Test Clock (TCK) pin 10-3, 11-8, 11-20
Test Data Input (TDI) pin 10-3, 11-8, 11-11, 11-20
Test Data Output (TDO) pin 11-9, 11-20
Test Flag true (TF) condition 3-54, 3-55
Test logic Reset ($\overline{\text{TRST}}$) pin 10-3, 11-9, 11-11, 11-20
Test Mode (TMODE) bit 10-11
Test Mode Select (TMS) pin 10-3, 11-8, 11-11, 11-20
Test, bit 2-21
Timed release bus mastership 7-99
Time-Division-Multiplexed (TDM) mode 1-15, 9-28, G-14
(*See also Serial port, multichannel operation*)
Timeout, bus mastership 7-108
Timer 1-11, 3-50
Timer and sequencing 3-50
Timer Count (TCOUNT) register 3-50, A-31
Timer Enable (TIMEN) bit 3-50, A-7
Timer enable and disable 3-51
Timer Expired (TIMEXP) pin 11-5, 11-18
Timer Expired High Priority (TMZHI) bit 3-51, A-19
Timer Expired Low Priority (TMZLI) bit 3-51, A-21
Timer Period (TPERIOD) register 3-50, A-31
Timing
External port 7-1
Host read/write cycles 7-62
Link port handshake 8-10
requirements 1-13
Synchronous transfers 7-64
TMS pin 10-3
Toggle, bit 2-21
token passing

link ports 8-17
Top-of-loop address 3-22
Top-of-PC stack 3-53
TPERIOD register 3-50, 3-51, 3-52
Transfer Control Block (TCB) 6-9, 6-71, G-14
Transfers (*See Short word, Normal word, Extended precision normal word, or Long word*)
Transfers through the EPBx buffers 7-121
Transmission errors, link port 8-16
transmission line impedance, link port 8-21
Transmit Clock (TCLKx) pins 9-3, 9-21, 11-7, 11-11
Transmit Clock Divisor (TCLKDIVx) bits 11-16, A-77
Transmit Count (TCNTx) registers A-78
Transmit data (TXx) registers 6-4, A-73
Transmit data Status (TXS) bits 6-63, 9-9, 9-38, A-73
Transmit Divisors (TDIVx) registers 9-3, 9-7, 9-14, 11-15, A-77
Transmit Frame Synch (TFSx) pins 9-3, 9-9, 9-22, 9-27, 9-30, 11-7
Transmit Frame Synch Divisor (TFSDIV) bits 9-15, A-78
Transmit Frame Synch Required (TFSR) bit 9-11, 9-22, A-72
Transmit serial data (TXx) registers 9-7, 9-9, 9-13, 9-17, 9-20, 9-27
Transmit sign extension 9-20
Transmit Underflow status (TUVF) bit 6-63, 9-9, 9-13, 9-27, A-73
Transmit/receive DMA, external port (TRAN) bit 6-15, A-55
Tristate versus three-state G-15
True always (TRUE) if condition 3-56
Truncate, rounding (TRUNC) bit 2-2, A-4
Two-dimensional DMA 6-10
  Link port 6-83
  Sequence 6-87
  Serial port 6-93
Two-Dimensional DMA Enable (D2DMA) bit 6-51, 6-53, 6-84, A-75
Twos-complement data 2-5, 2-9
Type, data (*See Data types*)

U

Unaligned 64-bit Memory Access (U64MA) bit A-14
Unaligned 64-bit Memory Access Enable (U64MAE) bit 5-29, 5-36, A-8
unaligned memory access A-8
unbanked memory 7-10
Underflow (*See Multiplier*)
Underflow exception 2-4
Unified address space 1-13

Universal (Ureg) registers 1-12, 2-38, 5-7, A-17, G-15
control and status A-2
data address generator A-32
processing element A-15
program sequencer A-17
Universal Asynchronous Receiver/Transmitters (UARTs) 9-5
Unpacked data 6-17, 7-8, 7-51, 7-69, 9-18, 9-37, 9-38
Unsigned
data 2-5
input 2-20
User-defined Status (USTATx) registers A-15
Using the cache 3-12

V

Values, saturation maximum 2-17
Vdd power, analog (AVDD) pins 11-9
Vdd power, external (VDDEXT) pins 11-9
Vdd power, internal (VDDINT) pins 11-9
Vector Interrupt address (VIRPT) register 3-49, 6-56, 7-81, 7-82, 7-124, A-47
Vector Interrupt Address (VIRPTA) bits A-48
Vector Interrupt Data optional (VIRPTD) bits A-48
Vector Interrupt Pending (VIPD) bit 7-126, A-52
Vector Interrupt, Multiprocessor (VIRPTI) bit A-19
Vector interrupt-driven messaging 7-82
Vector interrupts 1-14, 7-81, 7-124, G-11
Host 7-82
Interprocessor 7-81
VIPD bit 3-49
VIRPT interrupts 3-45
VisualDSP++ 1-16
Von Neumann architecture 5-2, G-15
Vss ground, analog (AGND) pins 11-9
Vss ground, digital (GND) pins 11-9

W

Waitstates 1-13, 5-36, 5-37, 7-14, 7-24, G-15
Waitstates and Access Mode (WAIT) register 6-4, A-48, A-49
Web site 1-23
Word rotations 5-23
Wrap around, buffer 4-9, 4-12, 4-16
Write High/Low ($\overline{\text{WRH/L}}$) pins 6-30, 7-6, 7-46, 7-61, 7-95, 7-105, 11-4, 11-11
Write, synchronous timing 7-64

X
xMODE bit 10-10
Xor, Logical 2-8

Z
Zero, round-to 2-4